附 属 資 料　　経情協47－2

コンピュータ情報システムの開発における

システム・ライフ・サイクルのマネジメント

昭 和 48 年 3 月

財団法人　日本経営情報開発協会

この報告書は昭和47年度における日本小型自動車振興会から小型自動車競走法に基づく小型自動車等機械工業振興資金の交付を受けて作成したものであります。

附属資料 1

情報システム開発サイクルのコントロール

これは Robert I. Benjamin著 "Control of the Information System Development Cycle" Wiley, 1971を全訳したものである。

まえがき

この本では、情報システムの開発を管理する為の概念的な方法論を開発することをねらっている。この方法論は、情報システムの開発を目指している大企業、政府機関あるいは小さいながら独立してシステム開発をしている組織体などの様々なタイプの組織体が、そのニードに合わせて容易に修正できるようにかなり一般的なレベルで述べられている。

この本では、「システム開発サイクルとは何か、そしてそれをいかに管理したらよいのか」というテーマだけにしぼることにして、このテーマの中である程度はふれることになるがそれだけでも1冊の本を構成しうるような、計画や開発されるシステムの選択に関する評価基準、データベースの構成あるいはそのような問題といった興味深いテーマは除外することにする。

この本の大まかな構成は次のようになっている。

1. 情報システムを開発する場合の環境に関する問題の記述。
2. システム開発と新製品開発のようなどこの組織にもあるような他の開発問題との関連。
3. 情報システムの開発サイクルを管理する時に基本となる一群の原則の提示。
4. 開発サイクルモデルの開発。
5. 開発サイクルを管理する為の一群のストラクチュアルスタンダード（structual standard）の提示。

この本が生まれるまでにはたくさんの手助けが必要であった。私のアイデアをまとめ、本にするように進めてくれたPaul Strassmann、さまざまな助言をしてくれたDan TeichrowとGeorge Langnas、この本の構成と

操作手順の開発に関する節での仕事をしてくれたMrs. Fern Mackour、タイプをしてくれたMiss. Barbaba Smithの方々に感謝している。また長い間、私のアイデアの多くを生み出すことに協力してきた私の助手達にも感謝している。

Robert Benjamin

ニューヨークにて

１９７０年１１月

表と図のリスト

目　　次

序　文

コンピューター技術の非常に急速な発展によって、現在のシステム開発部門に大きな圧力がかかってきている。システム開発部門が、その導入が要求されている大規模で複雑なシステムによる挑戦を受けて立つ為には、自ら基本的な変化をせざるを得なくなってきている。

下に、コンピューター技術の発展によって、システム開発担当のマネジャーになげかけられている問題の中のいくつかを示す。

1. ハードウェアとソフトウェアの両面でのおどろくべき技術的な進歩によって、経済的に実行可能であるアプリケーションの数がはげしく増加してきている。
2. プロジェクトのライフサイクルの期間が長くなってきている。
3. プロジェクトの予算は巨大なものになって、プロジェクトが完全に理解されないうちに承認されている。
4. 導入に成功したシステムが多くなるにつれて、情報システム活動の固定費が急速に上がってきている。
5. システムの専門家の不足が深刻になってきているが、解決の様子はみられない。
6. システムのユーザーとそのシステムを作り上げる技術的な専門家との間では、要求や問題についての意味のある連絡がほとんど行なわれていない。
7. 開発プロセスの概念については、開発を担当している組織の実行部門だけが今まで経験してきたものをおおまかに理解しているだけであって、ほとんどのコンピューターの導入に責任をもっている管理部門のマネジメントにとって、それは関係のないものと思われている。

これだけのおどろくべき数の障害があるにもかかわらず、システムマネジャーの責任となりつつある、より規模が大きくてより複雑なシステム開発プロジェクトに対するあくなき要求に対して、コンピューター革命が対処していくことを現在の組織体をとりまく経済状態は要求している。この困難な役割を果たす為には、このシステムマネジャーはシステム開発プロセスを管理することを学ばざるを得ない。彼は、プロセスを管理し、必要に応じてユーザーと技術者の両者に情報を提供することができるように、このシステム開発プロセスを理解しやすく、役立つ活動のレベルまでブレークダウンするべきである。

この本では、システムマネジャーが効果的に開発プロセスを管理できるように、システム活動を構造化（モデルとして）して述べることにする。最初に、システム活動に特異な問題とともにマネジメントの問題のどんな解決案にも含まれているようないくつかの一般的な原則にふれる。２番目に、システム開発を行なっているどんな組織にも適用しうるシステム開発プロセスのモデルを開発する。最後に、システム開発にあたっている組織内の要員をモデルのフレームワークの中で行動させるようなストラクチュアルスタンダード（あるいは標準手順）についてふれることにする。

この本は３つのクラスの読者に合うように書かれている。

1. コンピューターシステムの技術的な面をしらないユーザーで、彼がシステム開発サイクルの中にどのように参加すればよいのか知りたいと思っている人々。
2. 多くの場合、組織の中でも技術的ではない部門から来ていることが多いシステム開発を担当するマネジャー。
3. システム開発の専門的な分野で働いていて、システム開発プロセス全体

をもっとよく理解したいと思っている技術専門家達。

第1節　ストラクチュアル・スタンダード(structual standard)の概念

システムの開発プロセス

この文献で言わんとすることは、システムの開発プロセスが存在すること、そしてシステムの作成を成功させるにはその開発プロセスを理解する必要があるということである。システムの開発プロセスとはどのようなものなのだろうか？

第一に、それは、いわばある種のことがらを構築することである。建物を建設することは一種の開発プロセスの例である。

第二に、われわれが情報システムと呼んでいるものは1つの工場として考えられるということである。いわば、ボーイング747の開発サイクルのプロダクトがボーイング747をつくる工場システムを意味しているように、この情報システム工場は目に見える形に描き難いものである。単純な見方をすれば、情報あるいはデータ工場とはその組織のための計算設備を収容している場所といえよう。現実には、特定のシステムのためのデータ工場とはインプットを投入する人間、それをコンピューターが受け入れる形式に変換する人間、データの流れをコントロールする人間、個々の地点間のデータ伝送を可能にするコミュニケーション・ネットワーク、計算設備といったものを含むものである。

コンピューターを利用している組織の中でも、一般的な開発プロセスになじんでいる程度には差がある。一方の極には、新型の飛行機や自動車の開発にビジネスが強く方向づけられている飛行機や自動車の製造業がある。他の極には、保険会社、銀行といったような、開発サイクルに関係する業務内容に全く適応していないサービス産業が存在する。この中間には、新製品開発

に関係する小規模のグループだけが開発プロセスを理解しているような一群のマーケティング指向の会社がある。これらの組織では、コンピューターの管理責任は、開発サイクルになじんでいない管理者の管理下にあることが普通である。

ストラクチュアル・スタンダード

システム開発プロセスの業務にたずさわっている人々に、あるやり方に従って手順を考えださせたり、個々の手順のなかで代替案を考えださせたりする標準を定義することは、システム開発プロセスを管理するために用いられる最も重要な用具の一つである。われわれはこのような標準を"ストラクチュアル・スタンダード"と呼ぶことにする。

まずい標準故にシステムプロジェクトが失敗することはしばしばおこるものであり、しかも標準の内容と、解決すべき内容を理解している人々がごく限られていることがそのような結果をもたらすものである。例えば、開発プロセス、データ変換、プログラミング言語について標準が定義されてきている。ところが、このような標準の種類分けをすゝめてゆくのにあたり、システム部門の管理者が開発努力結果を維持しコントロールするために、あまりやる気を起こしていない彼の部下たちに標準を課すための理由を見出しにくいものである。

一般に標準はそれらの目的によって分類することが出来る。例えば製造工場の場合の、パーツの互換性を高めるための標準という意味でのネジの標準サイズはこの種の標準の典型的な具体例である。一方、システムの場合には、別の計算環境（computational environment）のもとでのプログラムの互換性を促進させるようなプログラミング言語や余裕をもたせる要素の選択、

あるいは、アプリケーションの間でのデータの互換性を促進させるためにデータ・ベースの記述をコントロールすることなどがその典型的なものである。

標準が1つ以上の目的をもちうるという事実はその分類をやりにくくすることになる。したがって、プログラミング言語の選択は、互換性に影響するのみならずシステムの効率にも作用を及ぼす。標準を使用するうえでのいくつかの混乱は、標準をつくりだす理由を調べることで取り除くことができる。表1－1には、標準の4つの存在理由をあげてそれらの間にある関係を、一般的な関係とシステムの面での関係に分けて記述してある。これら4つの標準分類の範疇をさらにひろげていくのにはもっと作業を積み重ねていかねばならないであろう。しかし、ここに示す範囲だけでもシステムに関する領域で標準を設定する場合に多くの混乱と抵抗が生じる理由を説明しているものと信じている。

その標準の個々について目的を明確にしていくだけでも組織で“標準”の売り込みをもっと改善していくことが可能になるであろう。

標準は、それらが何を達成しようとしているのかという面からみることによって使いやすく分類され、最善の理解をえることができる。

表1－1をみると、他の標準は、何事かをおこなうルールあるいは事実が起きた後で何事かがどの程度首尾よくおこなわれたかを測定するルールを提供するのに対し、ストラクチュアル・スタンダードは問題を解決するためのモデルに従ってゆく手続きを記述しているものであることがわかる。表1－1にあたえられているストラクチュアル・スタンダードの2つの例はこの点を明らかにしている。予算とは、組織の要求と組織のリソースのバランスを組織がいかにとっているかということと、一致した目標について積極的な業務遂行を組織がいかにひきおこさせるかということについてのモデルである。

ＰＥＲＴチャートは、特定の作業を完成させるのに必要なリソースや事象の流れを示したモデルを定義したものである。予算にしてもＰＥＲＴにしても、“なされるべきこと”を確認したうえで、事実が起こった後に“すでになされたこと”を評価するための測定ができる余地をもっている。

表１－１　標準の範疇

標準の理由	例　（一　般）	例　（シ　ス　テ　ム）
互　換　性	ネジの標準サイズ	データ変換用のＡＳＣＩＩコード
		ファイル変換用の磁気テープラベル
		システムの存在期間にわたって用いられるプログラム変換用のプログラミング言語
パーフォーマンス	建築コード（Building codes）	プログラミング言語やソフトウェアシステムのサブセットの可能案の選択
		売り掛け金額のような、システムのなかで肝じんな部分のバランスをとるためのコントロール標準
測　　定	販売分析と報告	プロジェクト会計システムの利用
		データセンターのジョブタイム計算システムの利用
構　造　化（structural）	予　算	ＰＥＲＴ　チャート

このようにストラクチュアル・スタンダードはいくつかの非常に有用な性質をもっている：(1)それらは問題に対する解答を、その解答のためのモデルを用いることによって導きだすための標準手続きであると同時に、(2)それらは解答の成功の度合いを測定できるものである。

ある種のケースでは、その測定は定性的な範囲のなかでしか出来ないこともある。例えば、設計プロセスの初期の段階で詳細な仕様を要求する標準では、成功した設計が完了しているか否かについては、定量的測定よりもむしろ定性的なものを可能にすることになる。

要約すると、ストラクチュアル・スタンダードは、システム開発活動のなかで訓練によってひき出される独創力をとりだすための用具として非常に重要なものである。第4節から第9節で、システム開発プロセスのための標準が記述される。この基本ストラクチュアル・スタンダードは、開発プロセスのなかで十分な数のチェック・ポイント（あるいは作業フェーズ）を定義するものであり、そこでは開発プロセスが管理の状態のもとにおかれているか否かを決めるための測定や評価が出来るような構造化されたアウトプットを個々の標準がもっている。さらに、個々の作業フェーズに使いやすく細分化されている。それぞれの論理的作業フェーズに対して、1つのストラクチュアル・スタンダードが、活動がとどこおりなく進行実行され、評価されることを保証するために案出されることが可能である。

第2節　システム開発マネジャーが直面する問題

概　要

ピーター・ドラッカーは工業社会と知識生産社会を結びつける過渡的な世界を詳細に描きだしている：

「前世紀においては手作業を能率的に遂行させることがマネジメントに課せられた主要な仕事であったのと同様に、今世紀では知的作業の生産性を高めることがマネジメントの主要な仕事になるであろう。生産性のもとに管理された知的作業と管理されずに放置されている知的作業の間にみられるギャップは、科学的管理法が導入される前と後の手作業の間にみられる著しい相違よりもおそらく、さらに大きなものになるであろう。」

システム開発は産業における知的作業の広範な適用のあらわれのなかの一つであり、われわれはそれを管理する方法を真に知っているわけではない。ストラクチュアル・スタンダード（あるいはモデル）はシステム開発プロセスを管理する助けとなるものである。本節では産業がシステムプロジェクトを効果的に管理する能力をもっていない背景に横たわっているいくつかの要因（それらは知的作業のなかに一般的問題として含まれているものである）をしらべることからはじめることにする。

爆発的な技術進歩

コンピューターにまつわる技術の急激な進展は、増加的に能力を拡大していく技術的な発明の進展とそれを受け入れていくことの両面で目をみはるものがある。われわれの社会体制はコンピューター革命を、他にみられるいずれの全く新しい技術と比べてみてもよりはやいペースで受け入れている。コ

ンピューターの分野のハードウェア面でなされる発明は、市場性の面からの制約を受けているだけであり、この制約のなかにあっても、コアメモリーサイズ、サイクルタイム、ランダムアクセス容量といった主要性能に関するコスト・パーフォーマンスはどの5年間をみてもかなりの勢いで一般に増加している。ソフトウェアの面では、新しい特殊領域がたえず出現しておりそこでの発明のペースは大へんなものである。一つの例を考えてみよう。およそ15年間の短い期間のなかで、プログラミング言語の全体の領域にわたってつぎにあげるような種々の一連の革命が進展しその成果をわれわれは享受してきた。それらはサブルーチン、プログラム・アセンブラー、マクロアセンブラー、手続向きのコンパイラー、ファイル管理（あるいはシステム向きの）コンパイラーなどである。そのうえさらに、オペレーティングシステムを定義し作成する言語を研究する領域では非常に意義のある仕事がおこなわれてきている。

実施上のコスト／パーフォーマンスの点で個々にかなりの規模の改善がなされた結果、以前には解決不可能であった問題の解決が可能になったといえよう。ベル電話研究所のハミング（Hamming）博士がつぎのような仮定的状況を指摘したことがある。すなわち、もし自動車のコストがコンピューター・コストと同じ動きを同時にしたとすれば、駐車場を確保することが困難な状況のもとでは、自動車を乗りすてにし必要に応じて別のものを購入することになるだろう。われわれは、成長曲線はやがて横ばいになるにしても現在のところこのような終局の状況を考慮してものごとを計画することはないという点を認識すべきである。

システム開発が進められる環境にみられる技術的成長の動向を要約した広く引用されているつぎのような意見がある、「数年にわたってむやみやたら

になされた成長の結果として、コンピューターの分野はやっとその幼年期に近づきつつあるようにみえる。」

このようにして、ハードウェアとソフトウェア技術が向上するにつれ、成長性のある正統的なアプリケーションの領域もまた成長率が長期的に維持され増加することになる。そしてこの成長率に対する抑制要因としてあらわれてくるものは利用可能なリソースと新しいアプリケーション・プロジェクトを管理する能力である。

訓練された要員の不足

システム専門家の著しい不足が存在し、将来にも明るい材料がみられない事実が特に重要だというわけではない。その事実からもたらされる影響が非常に大きな意味をもつわけである。

1. 業務の遂行を保証する技術上あるいは管理上必要なスキルをまさにもちあわせていないグループによって、大規模なプロジェクトが試みられることになること。
2. 大規模なプロジェクトを通じての継続的な人事が、これは非常に好ましいことなのだが、つぎのような理由によって片すみに追いやられてしまうこと。つまり、他人の目にとまるようなスキルをもった人々が短期間のうちにいっそう責務の重い別のプロジェクトの業務に昇進していったり、他のどこかの組織のなかによりやりがいと責任のある職務を見つけだしたりするといったことである。人手不足によって、人々は彼らが能力をもちあわせていない段階の仕事に昇進してゆくことになるというピーターの法則をいっそう助長させることになる。

この問題を軽減するには、つぎの2つのことが必要となるであろう。それ

はシステム開発プロセスのなかにいる専門技術者に必要な教育のタイプを理解することと、教育はいくらかは、(1)システム開発者、(2)企業が提供するコンピューターの専門教育、(3)大学によって最善のものが出来るという点を認識することである。多量の必要な教育は要求にあった教育を開発していくなかで努力の二重手間を求めるものではない。

ユーザーが要求するコミュニケーション

一般につぎの点が問題なのである。

"ユーザーは、コンピューターの専門家が使う言葉で彼の要求を示せないこと、そしてコンピューターの専門家はユーザーの要求を彼が理解したとき、ユーザーの用いる言葉や用語で言いなおすとは限らないということである。"

ここに述べたことが本当だとすれば、開発サイクルの最終段階で具体化されるシステムはユーザーが望み、期待しているものと比べてかなり偏したものになるわけである。こゝで必要なことは、ユーザーと技術専門家の双方に理解出来る言葉で仕様が記述されることである。

この問題にみられる他の側面は、システムの活動が、そのシステムが貢献するはずの中心的組織目的（例えば新製品の導入とかセールス・キャンペーンなど）から分離されることである。システム活動はサービス機能であり、その利用者は組織のなかでの実行部門（生産、販売といったような）であり、この事情からして何がビジネスの要求事項であるかを判断することはシステム活動にとって困難になる。このことをうまくやるには利用者との間に有効な対話をもたねばならないのである。

プロジェクト・スケールの増大

プロジェクトは開発の時間や必要とするリソースの面で大規模になってきている。大組織で実現される小規模の良く定義されているシステムでは、初期の計画から最終的に導入されるまで1～2年を要するであろう。大規模の良く定義されているシステムは、それが完了するまで3年はかかることになろうし、大規模な開発的プロジェクト予算も伸びており、民間企業において開発予算が100万ドルを越えるシステム・プロジェクトをよくみかけるようになってきている。

状況をさらに混乱させることは、小規模システムから大規模システムになるに応じて導入に要するリソースの量が大きくなることである。図2－1はシステムの規模に対する人的資源の投入の関係を示したものである。少なめに見積った限りでも、システムの複雑さはシステム規模に比例して一般に増大するとしても、その増大に応じてリソースの必要量は幾何級数的に増すといえる。

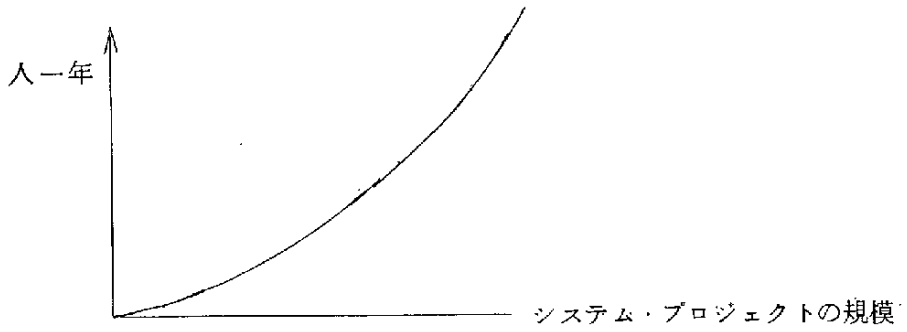


図2－1　システム規模と人的資源投入との関係

たしかに、長期間にわたって巨額の資金を投資することはすべてのシステム・プロジェクトにつきまとっている基本的なリスクを増大させる。

すなわち、

1. 開発の真のねらいはユーザーのニーズに一致しているか？
2. システム仕様のフェーズでユーザーのニーズを充足していたシステムでも、導入時に依然としてユーザーニーズを満たすだろうか？
3. 実働にたえるシステムが本当につくられるのか？

大規模化してゆくシステムに投入される努力に対して加えられてくる圧力にはつぎのようなものがある。

1. 組織の地域的、機能的境界を横切る標準化されたシステムに対するマネジメントの欲求
2. 計算処理力を集中化することによって生じるコスト・パーフォーマンス上の利益
3. 同一か類似のシステム開発努力にみられるコスト面の無駄の排除

しかし、これらの圧力はシステムに投入される努力の規模に対して注意深く重みづけしていく必要がある。というのは、システムに投入される努力は組織のキャパシティーを越えやすいからである。

固定費の急速な増大

それぞれのシステム導入は、全体的な企業活動として固定的費用を発生させる。すなわち、いったんシステムが開発サイクルをすゝみはじめると、それはなんらかの資本投資と同じ道を歩むわけで大きな固定的な操業費を生み出すのである。さらに、これらのシステムは組織の主要な情報の流れを処理しているのでそう容易に取り除くことは出来ない。もし、システムの初期計画の段階で全体の企業活動の面から見て妥当とされるコストを見積らない場合、そこから生じる固定費の内容を管理することは不可能となる。

研究・開発への指向性

現代の産業組織では、主に生産、マーケティング、流通、財務会計において組織の測定のために専門知識が必要とされる。そんなわけで、産業組織は本来の性質からしても競争的性質からしても短期的視野のもとでの結果に対して生産を実施することのくり返しを指向している。しかし、システムの活動は、1回限りの開発プロセスについての長期的方針をもたねばならない。産業上の環境のもとではきのうの結果は今日の行動を左右するのに対してシステム上の環境のもとでは、個々のシステムは大いに独立して存在しているといえる。過去のシステムがそれ以後のシステムにいくらかの専門知識や一般的知識の形で貢献するということはほとんどないわけである。一般に、システム活動に要求される主要な力はシステムをとり込んでいる組織の多くの部分に関係したものでないといえる。

システム専門家は、他の機能集団の場合にも言えることだが、彼の組織目的に反する目的のもとで仕事しているようにみえる。彼は自分の腕が許す限り最良のシステムを制作したがる。彼のマネジャーは日程と予算に従おうとする。開発しようとしているシステムに対するこれらの矛盾した接近の仕方は、常識的で相互が満足するやり方で妥協点に達しなければならない。同様に、コンピューターの技術者は、新しくかつ彼自身の失敗によって進歩を感じるようなものを開発したがる。これもまた、時間通りにおこなうことで満足をえるという彼のマネジャーの目的に反する目的のようにみえる。

ま と め

従来は、システム開発プロセスの合理的なアプローチを開発するのには、あまりにも時間的余裕がなく、システム活動に加えられる効率向上を促す圧

力がなさすぎたといえよう。システムに投入される努力の規模が増大するにつれ、管理のための明確に定義された構造なくして大規模のシステムに努力を傾けることは無駄を多くすることになる。先に示した種々の問題に初めのうちに直面した、最も規模の大きいシステムの開発者（すなわち、政府機関やコンピューター・メーカー）が開発プロセスのモデルを明確にしたり使用するうえで主導的位置をしめてきていることは不思議なことではない。

文献にはシステム・プロセスに関係して参照すべき部分や、そのプロセスを記述したりコントロールするための手法が豊富にもられている。しかし、成功した話を多くみることは出来ない。システム開発というのは困難な事業であるということが教訓である（事実はそれよりもきびしいが）、そしてわれわれがそのことを理解すれば、そのプロセスを切りぬけて働きのあるシステムを作成することになるであろう。さらに、われわれのアプローチを洗練すればそれ相応の効率を達成する点に到達することができるだろう。

第3節　システム開発プロセスを管理する際の基本的原則

概　要

前節では、システム開発プロセスに関連した問題のいくつかにふれた。この節では、システムプロジェクトを管理する為の広い意味での原則について述べることにする。まず、新製品開発サイクル（これは多くの組織で比較的よく行なわれている）とシステム開発サイクルの類似点を検討する。続いて、マネジメントがこのプロセスの管理に成功する為に、開発サイクルの中にもり込まれていなければならない原則を述べることにする。

システム開発サイクルと新製品開発サイクルとの関係

システム開発は、システム開発と新製品開発の両者の間での類似（表3－1）をみることによって、より容易に理解することができる。

新製品開発サイクルについては、ほとんどの会社でよい公式化がなされている。これには、組織内での限られたリソースを有効に使う為に、チェックポイントが組織的に如何にチェックを行ない、バランスをとるかを明確にした複雑なコントロールシステムが含まれている。新製品サイクルは、マーケティングを担当している者の目にひらめいた市場での成功のきらめきをもとにして、新製品の全般にわたる開発をするという意味で、確かに主要なストラクチュアルスタンダードである。システム開発サイクルは情報システムにおいても、これと同じように完成されるべきである。

しかしながら、開発プロセスを十分に理解している会社でさえも、とんでもないものを作り出してしまうことがしばしばある。

新製品開発	システム開発
1. 市場の要求の明確化。 2. その製品を、満足すべき利益のもとで開発し、市場化することができるかどうかが、財政的な分析と工学的分析によって確かめられる。 3. 概念的な仕様の決定から始まってそれぞれの要素の詳細な仕様決定で終わる設計工学（design engineering）の知識が製品を製作するのに必要である。その製品仕様での市場パフォーマンス並びに財政的な目的に対するその製品のパフォーマンスが吟味される。個々の吟味の時点において、プロジェクトは続行されることになったり、修正されたり、中止されたりする。この場合、製品の原型（prototype）が作られる。	1. ユーザーのニードの明確化。 2. 限られた開発リソースをうまく分配することによって、システムに対する支出が経済的に妥当であり、システムが技術的に実行可能であるかどうかが、コスト対利益分析によって検討される。 3. 情報システムの設計が行なわれる。一般的でかつ概念的なシステム仕様がシステムを完全に記述し、文書化することのできる細かさをもつまでレベルダウンされる。そして、システムをユーザーの要求や財政的な目的と、比較しながら吟味することによって、それを続けたり、修正したり、中止したりする。 もしプロジェクトが革新的なものであれば、システム要素の原型がいくつか実際にプログラムされ、テストされる。設計が現実的であることに同意する前に、例題によって使われている予測手法が将来のある状況を

適切に予測していることが確かめられる。

4. 製造工学（manufacturing engineering）によって製品がいかに製造されるかが決定される。これによって、製品の作られる仕事の順序や製品を作るときの要求事項が定義される。そしてここでコストと仕様がもう一度評価され、製品開発を続行するか、放棄するか、修正するかの決定がなされる。

5. このステップまでで、製品をいかに製作するかが理解され、必要となる道具や治工具やダイス（dies）が設計されるべきである。

6. ここで、必要となる道具が設計される。

7. 製品が、実際の製造環境の中で設計仕様を満たすことができるかどうかをみる為に、パイロットプラントでのテストが行なわれる。

4. 技術的な設計工学が適用される。これはインプットがいかに希望のアウトプットに変換されるかを調べるステップである。事象の順序、システムの流れ、必要となるファイル、そしてプログラムと手続きなどが明確にされる。さらに細かい情報をもとにして、プロジェクトの続行、放棄、あるいは修正の決定がなされる。

5. この段階になると、システムを開発する為の技術が理解され、プログラムと手続きが設計されるべきである。

6. プログラムがコーディングされる。そしてユーザーが理解することのできる形で手続書が書かれる。

7. プログラムと手続きが、工場（ユーザーとデータセンターのあるところ）の環境の中で設計仕様に従って働くかどうかを見る為に、システムとしてテストされる。

8. 製品が組立ラインで製作されている。	8. システムはその組立ライン（ユーザーの手続きに従った行動とデーターセンターのコンピューター稼動の組合せを示す）から、今や情報を生み出している。

表3－1　システム開発と新製品開発の比較

システム開発の為の原則

成功したシステム開発プロジェクトに共通している基本的な原則がいくつかある。この節の残りでは、我々が原則（axiom）と呼ぶこれらの原則について述べることにする。そして4節以降の各節では、これらの原則をシステム開発サイクルのスタンダードとしていかにまとめあげるかについてふれることにする。

原則1　情報システムは1つの資本投資である

情報システムは資本投資のことであり（財政を担当するマネジメントがそれをどのように取り扱うかにかかわらず）、他のプロジェクトと同じようにコストが評価されるべきである。これが基本である。

組織体が技術への投資としてコンピューターを必要とする時、あるいは本社に備えてあることが普通である機械の1つとしてコンピューターを持とうとすることがある。

しかしながら、今日ほとんどの企業はコンピューターを持っており、そこで様々なシステムが稼動している。それでコンピューターの神秘性は徐々に消えているようである。コンピューター技術におけるコストパフォーマンス

が増加してくるとともに、利益を生みそうなアプリケーションの領域がシステム部門によって開発されるのを待っている割合が急激にましてきている。開発を規制する唯一の要因は、少ない人的リソースである。

コストを正当化するという規律を設けることによって、システム部門はその努力を、組織にもっとも貢献するアプリケーションの領域に集中するし、ある分野へのアプローチがもっとも良いと判定されるまでは他の分野へのアプローチも行なうことになる。そして最も良いアプリケーションを選ぶという方法をとれば、プロジェクトの選択に用いる基礎資料をマネジメントに提供することができるわけである。マネジメントは、システム開発プロジェクトの資料の他にも、他の分野でのプロジェクトの選択についての基礎資料を持っている。（例えば人事情報システムからすぐに見返りが得られるわけではないけれども、このシステムはその組織体の構造上での変化に関する計画を担当しているマネジメントにとっては基本となるものであるし、従業員の能力を知ることもまた必要なことである。）

原則2　システム稼動の成功度合を測定する評価基準を作れ

システムプロセスの中で、そのシステムを開発すべきかどうかを決定する為に、システムの価値を経済的に評価することは必要である。一般的には、経済的な要求が、システムの成功尺度（success criteria）と呼ばれているものの一部分になっていることが多い。

例えば、従業員の仕事の経歴、教育、スキルあるいはそのようなものに関する記録を作り出し、保全している人事情報システムは標準的な報告書を作成する他に、一時的な報告書を出したり、いろいろな目的に従ってシステムの中から様々な情報を探し出したりすることができる。たとえば、人事担当の副社長にこのシステムの成功尺度が何であるかを尋ねたと仮定しよう。す

ると彼の答えは次のうちのいずれかもしくはすべてであろう。

1. システムは、前もって決められたコストを時間の制約の中で、仕様書に記されている機能を達成すること。
2. システムの年間、従業員一人当りのコストは5ドルを越えないこと。
3. 従業員1人分の記録を修正する為にかかるコストは2ドル以下であること。
4. この情報を持った結果として、人事管理機能を遂行する為の総コストと組織体内での総コストの現在の比率を維持できるかあるいは低下させうること。

システムデザイナーは、システム報告書の中でこの副社長に彼の評価基準に従った場合に、システム開発が成功したかどうかを知らせることができる。システムの報告機構が直接に答えることのできない唯一の評価基準は上で述べた1である（実行されるように設計された機能が実行されたかどうかという点に関して不明確になるのである。）。ただ、直接に答えることはできないけれども、これらは他の質問への答によって間接的には答えられる。

システムそれ自体によって測定できるいくつかの成功尺度を典型的なシステムについて述べることにする。

1. 受注処理システム（order processing system）は、その設計の一部分として、次のような種類の統計を作り出すべきである。(a)1枚の請求書（invoice）を処理する為のシステムの平均コスト、(b)システムでの、年間、顧客当りの平均コストのパーセンテージ。同じような数字がシステムに含まれている配送機能（distribution）、伝票処理機能（billing function）、報告機能（reporting）の3つの基本的

な機能についてもはじき出されるべきである。このような種類の統計は、システムがいかに稼動しているかあるいはどこを改善したらいいのかといったことを明示する。稼動システムのある部分的にでもより妥当的であると思われるならば、この改善をすることによる成功の度合はシステムの中で測定されるべきである。例えば、受注システム(order entry system)において、顧客への配達サイクルを1週間から3日にすることが提案されたとしたとき、この機能がいかに実行されたかを測定する統計をシステムが作り出すべきである。

2. 給料システムは、チェック(check)当りのコストあるいは年間従業員1人当りのコストにもとづいて、その成功度合測定することができる。このシステムの実現性は、ADP社(Atomatic Data Processing Inc.)が提供している商用ベースの給料アプリケーションのように、個々の値段のわかっている様々な処理方法(option)にもとづいて調べられる。

3. 在庫管理システムの成功は、在庫の平均日数、売上げに対する在庫の比率、在庫スペースの減少、あるいはこれらと似たようなもので評価することができる。

いずれの場合においても、システムの成功度を示すいくつかのインデックスを設定することができるし、それゆえに元々の仕様書にもとづいて同意を得ることができる。これらのインデックスは、システムが稼動中でも、その機能を達成するのに成功しているかどうかを見ることのできるように、システム設計の中に組み込むことができる。

まとめていうならば、その飛行性能を測定する能力をもたない飛行機に乗りたくないのと同じように、それ自身の性能を測定しないようなシステ

ムは作るべきでない。それゆえに、システムの成功尺度はそのシステムのユーザーによって定義され、これらの評価基準の多くが、システムに組み込まれているメカニズムによって測定なれるようなシステムを設計すべきである。

原則3　システムはユーザーの為のもの

この原則について述べる前に、ユーザーとはだれであるかについてふれることは必要である。例えば、コンピューターの運営とシステム活動が管理部門(controller's department)だけによって援助されるような時に、この概念はあいまいになりがちである。というのは、この場合にはユーザーとシステム部門が同じだからである。しかしユーザーを分離するということはここでも重要なことである。例えば、給与部門、会計と財政の機能あるいは売上集計部門は普通この管理部門に対する報告義務はもってはいるけれども、彼らはシステムに対して独自の要求事項や興味をもっているのである。それゆえ、これらのアプリケーションの分野には、それが開発に成功する為には認識されなければならない個々のユーザーの見方というものがあるはずである。

これまではコンピューターの運営とシステム活動が組織のある2つの活動領域に位置づけられていることが多かったが、今や急激に組織全体に対するサービス機能となりつゝある。これはシステム構成が経済的なサイズという利益をうるようにコストパフォーマンスの考え方が修正されたことと、アプリケーション分野の間での関係の2つの原因によっている。ウィシントン(Withington)はこの急激な傾向をもつようになった主要な3つの理由をあげている。(1)標準化に対するマネジメントの要求、(2)人間の不足、(3)アプリケーションの複雑性の増加。この傾向はこれからもさらに増していくもの

を予想される。

システムグループの技術的な力が増してくるとともに、ユーザーのニードがある方向に向かっているにもかかわらず、彼らはジェネラリスト的なアプローチをとるようになって、システム優越感（system ethnoce ntrism）と呼ばれている大変な危険におちいる。これはシステムの設計にあたって、ユーザーの見地に立つのではなくて、システム部門の立場に立ってしまうという傾向である。それゆえにこの原則3の重要性はましてくる。

この原則は、確かに明らかではあるが、次のような事実がしばしば見落されがちである。

1. システムの経済面での妥当性を検討するのはユーザーの責任である。それゆえに、プロジェクトの最初からコストの妥当性を検討する過程ではユーザーが関係することが要求される。これがユーザーのプロジェクト参加への第一歩であり、プロジェクトを成功させて終わらせる為の重要な要素である。
2. 理想的にいえば、ユーザーは自分のシステムに関する仕様書の開発をすべきである。しかしこれは困難なことである。というのは、仕様書が局所的になる（すなわち、考えられるすべての新しいシステムを表わしているとはいえない）大きな危険がともなうし、プログラム設計を完成させるのに必要な詳細にわたる情報を含ませることはできないからである。普通とられる両者にとって満足できる折衷案は、ユーザーの要求に見合うような仕様書をユーザーのわかる言葉でシステム部門が作り出すことである。

   このような方法で開発された仕様書はユーザーの要求にも、またシステム部門の要求にも合ったものになる。

3. ユーザーは、仕様の改善のたびにすべての仕様書の変更に対して承認をすべきである。

4. システムを設置し、人員を教育し、データーをまとめ、そして導入の時期をきめる責任はユーザーにある。これを行なわないと、システムの導入がどんなにうまくやっても非常に長くかかることになる。

5. システムはユーザー自身によって導入されることになるから、理想を言えば手続書はユーザーのスタッフによって書かれることが望ましい。このこともまた現実的でない。というのは、手続書を書くことは急速に技術的に専門化しているし、手続書というのは、たぶんユーザーとシステムスタッフの共同の仕事として達成されることになるからである。

ユーザーの参加はシステムの導入を成功させる為にはすべての分野で必要であり、ユーザーの組織がどこまで参加するかは、問題の性質やユーザーの組織がＥＤＰの技術をどの位もっているかによって変わってくるものであるということは認識されるべきである。ユーザーが多くのシステムを提案すればするほど、参加の度合が増してくるのは、システム部門が普通認めるところである。

原則4　ランドマーク（Landmark ）を設定せよ

開発の努力を管理する為には、いつでもそれまでに何が終わっていて、いまどこにいて、何がなされているかを知ることは必要である。多くのシステムプロジェクトは相対的に異なっているので、適切なランドマークを選択するのはむずかしい。それらが役立つ為にはわかり易いものでなければならないし仕事の概念づけの助けとなるべきであるし、それらは定義のはっきりしたアウトプットである必要がある。

広い意味で言えば、システムプロジェクトは3つの主要な開発段階を経過

していく。システムが定義され、作成され、稼動される（すなわち導入されるということ）という3つの段階である。開発を終えると、システムはビジネスの実施部分の1つになる。このフェーズはしばしば保全のフェーズと呼ばれる。

定義のフェーズでは、システムのユーザーを満足させることのできる機能が概念的にとらえられ、これらの機能をコストが妥当であると認められた枠内で完成させることのできる技術的な設計が行なわれる。

作成のフェーズでは、システムがプログラム化され、手続化され、文書化され、そしてテストされる。

しばしば導入と呼ばれる稼動フェーズでは、ファイルや手続きが新システムのものへと移行される。システムの設計に対する誤解をもたらすようなシステムの変更、あるいは大量のデータを処理することによって得られる経験の獲得はこのフェーズで行なわれるべきものである。

ひとたび主要なランドマークがうちたてられると、それらはいくつかの役立つ管理の要素としてのランドマークに分割される。例えば、作成のフェーズにおいては、プログラムは個々に1つの単位として設計され、コーディングされ、テストされる。これらの要素の1つ1つがこのフェーズにおけるランドマークということになる。

開発プロセスは、ある時には一線に並ぶべきであるような多くの平行して進められる活動からなっている。重要なランドマークは、システムプロセスにおいて平行に進められている努力を一つに結びつけるものである。例えば開発プロセスのさまざまな時点において、プログラム設計、手続き設計あるいは移行計画の領域などは、比較的独立に仕事が進められている。それゆえ、かんじんなチェックポイントにくるたびに、これらの努力が調整され、1つ

のトータルデザインへと統合化されるべきである。

原則5　プロジェクトの中止を恐れるな

システムプロジェクトを、ステップに分けられた開発サイクルにブレークダウンすることによるもっとも大きな利点は、開発ステップのいずれの終了時点でもプロジェクトを延期したり、中止したりすることが可能になることである。図3－1は、開発サイクルでのコストと時間の消費についての合理的な範囲を示している。これによれば、プロジェクトをその段階1が終了した時点で中止するのは、経済的にもさして痛手とはいえないが、いったん、あるプロジェクトに段階2（作成のフェーズ）に進む許可が与えられたとすると、その金額の支出はきわめて大きなものになるであろう。

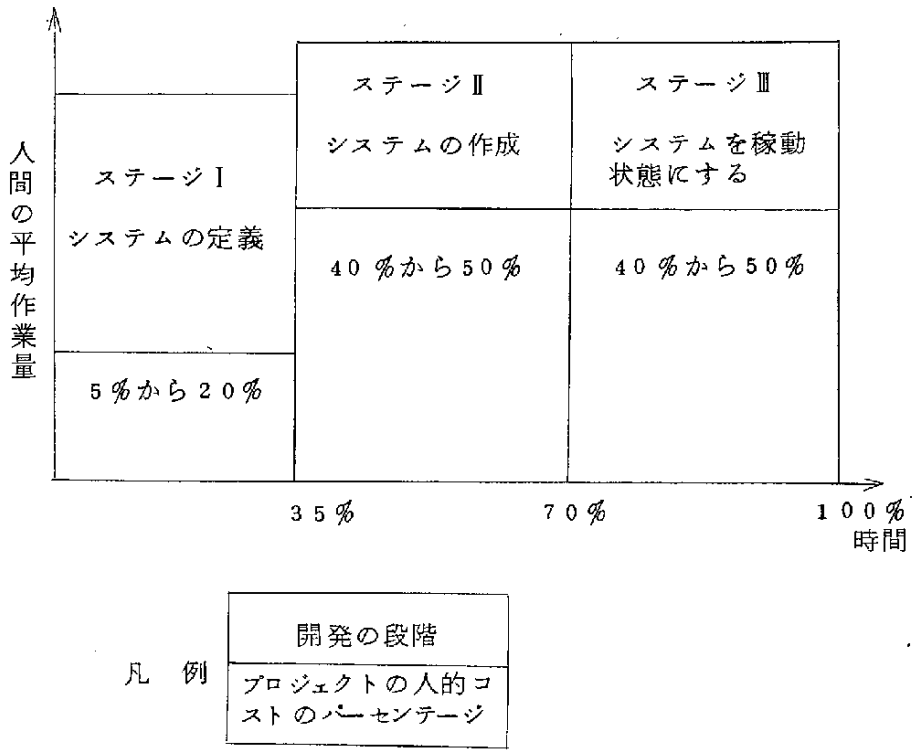


図3－1　システム開発サイクルでの時間とマンパワーの必要量

開発サイクルのもう1つの見方は、その現金の支出と失敗の危険という観点である。図3－2をみると、やはり定義のステップが決定的であることがわかる。このステップで重大な決定がなされることになる。この図をみれば、いくつかの点についても指摘をすることができる。

まず第1には、もっとも才能があって創造的な人間が開発プロセスの初期に要求されること。第2に、定義のプロセスを短期間で切り上げる割合は少ない。というのはもし省略すれば、開発を担当する者はいずれ非常に多大なコスト増加を覚悟して、もう一度もどってこなくてはならなくなるからである。第3には、ある程度の利害関係をもっているものはすべて、このシステム定義承認のサイクルに参加すべきである。というのは、プロジェクトに含まれる範囲を変えることができるのはこの段階だからである。この変更はマネジメントの目標に合致させるという観点にたつことによって、もっとも容易に影響を与えることができる。

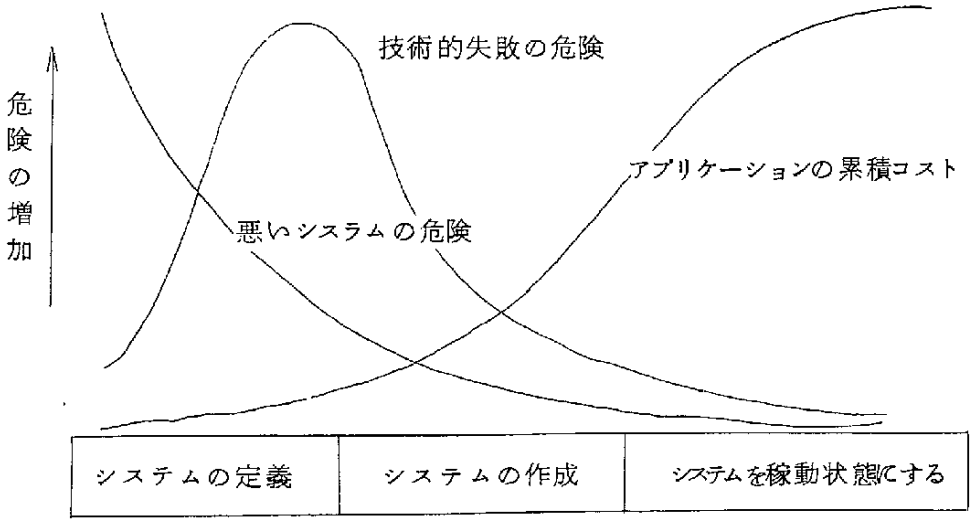


図3－2　コンピューターアプリケーションの危険とコスト

原則6　関連するところでは、マネジメントを参加させる

解決されねばならない基本的な問題が2つある。(1)組織にもっとも大きな利益をもたらすプロジェクトが選択され、明確にされるべきである。(2)選択されたプロジェクトは効果的に達成されるべきである。

プロジェクトを選択し、その仕様を決定するのは、開発サイクルの中の定義のフェーズで遂行されることになる。システム活動が、その活動の行なわれている組織のゴールに敏感に反応を示すのはこの時点しかないのである。

システム活動によって助けられるマネジメントは、定義のフェーズにあるプロジェクトを注意深く吟味することによって、システム活動における、そのマネジメントのもっとも重要な責任を遂行することができる。この時点でならば、システム活動の方向を再び設定しなおすことができる。いったん仕様が承認されると、その結果、非常に高い開発コストを支出する約束をするとともに、ビジネスを行なっていく上で固定費になるであろうシステム稼動の費用を認めることになる。

適切な吟味をする為には、システムを使うユーザーと利用可能な少ないリソースを最終的に分配すべき立場にいるマネジメントの両者が、精密な検討と評価が行なえるように、定義のフェーズで作成された仕様書が設計されていることを、システム活動の中で確認すべきである。

プロジェクトの技術的なフェーズがいったん始まってしまうと、マネジメントの管理が効果的に行なえるのは、そのプロジェクトがめんどうなことになっていることを知らせる早期警戒システム（early warning system）に限られてしまうことになる。これを行なうための標準的な手法はたくさんある。バートチャート、プロジェクト管理のネットワーク、月毎の開発予算の推定などが例としてあげられる。技術的な開発フェーズを監視することは

マネジメントの関心をもつところであるけれども、それは、特定の組織にもっとも適しているプロジェクトを選ぶことほどむずかしいものではない。

原則7　人間が問題である

知識を必要とする作業の特色は、製品を開発するときに関する投資に比べて極めて高いということである。これはどんな開発活動においても普遍的に真実であるばかりでなく、特に情報システムの開発においては明らかである。次に示すのは、情報システムと新製品開発サイクルにおける人間と設備の関係を比較したものである。

	新製品開発	情報システム開発
定義のフェーズ	ほとんど人間だけ	ほとんど人間だけ
作成のフェーズ	人間よりも設備により重きがおかれる	ほとんど人間だけ
稼動のフェーズ	主として設備が中心になる	主として設備が中心に

情報システム開発活動の稼動段階では主として設備が中心となってはいるが、インプットの移行に関しては相変らず人間が主要な役割を果している。このようにして、人間の問題がシステムの開発とその稼動の両者に関連していることがわかる。

プロジェクトのスタッフである開発担当マネージャー（development manager）は不適任者を最小にするという問題に当面している。もし彼が人事システム（pesonnel system）を開発しようとすれば、彼の部下であるプロジェクトマネジャーは例えば次のような特性をもっていなければなら

ない。

1. システム開発プロジェクトを管理した経験がある。
2. 人事システムの技術（ art ）の事情を知っている。
3. 組織内での人事の実情を知っている。
4. 人事システムに役立ちそうなハードウェアとソフトウェアについての知識をもっている。

そして彼は次のような意見をもっている。

1. 非常に疑い深い見方をするルネッサンス人（ Renaissance man ）を見つけ出す。
2. システムプロジェクトを管理した経験をもった人間を見い出し、上にのべた4つのカテゴリーについての十分な知識を彼に与える為に、勉強時間を確保してやり、それを助ける意味で、相談相手になってやる。もちろんこれは長くかかることになるであろう。というのは、ある適当なレベルに達するまでは、いい仕事がなされることは期待できないからである。
3. システム開発プロジェクトを管理した経験をもった人間を中心としたチームを作る。彼らは共同して働くということを学ぶ時間を必要とするので、チームが働き出すには時間がかかるであろう。けれどもチームのメンバーから引き出される専門的な知識は、開発されるシステムをすばらしいものにするはずである。
4. プロジェクトをあやまって管理する。

多くのプロジェクトは確かに管理を誤まっている。というのはここで述べた2とか3とかの方式を採用していないからである。大規模なプロジェクトチームによるアプローチは好ましい方法である。プロジェクトが小さくなればなるほど、プロジェクトチームの管理というものの重みが小なく

なるし、情報のニードに応じてプロジェクトマネージャー自身が必要とする知識を開発すればすんでしまう。

システム開発担当のマネージャーがもっとも関心をもつというのは、現在進行中のプロジェクトを成功させるのに必要な経験レベルではなくて、十分な計画をたてるのに要求される経験レベルである。

原則8　システム開発はくり返しの過程である。

大規模で複雑なプロセスを理解することは極めてむづかしい。まず第1に全体的な姿（big picture）を見てそれを理解すべきであり、それから、それを完全に理解できるまで細かくきり下げていけばよいことは、本能的にわかることである。

前で述べた学習にたとえられるシステム設計においてもやはり全体的な姿がまず描かれる。それから、プログラミングレベルでの完全な仕様書が書けるところまで段々に細かくしていって定義されるべきである。システムデザイナーは、概念的なレベルから非常に細かいレベルにいたるまでのシステム開発においては、数多くのくり返しが行なわれることを認識する必要がある。

個々のくり返しを区別することは、段階が進むにつれて徐々にその数が増してくる次の設計段階に参加する専門家達にシステムの内容を教えるのに役立つシステムの概念的な記述書を書くのにもまた便利である。設計のくり返しの数が合理的に最小であると思えるような開発サイクルを設定するということは、十分に考慮されたシステムを開発するのに大いに効果がある。

原則9　実行可能性のある設計の代替案をできるだけたくさん考える

正確に文書化されたくり返しのプロセスは、システムの最終的な設計が初期の設計案にある改善をほどこしたものになっているといういくらかの確信を与える。ある意味で、個々のくり返しは1つの設計代替案として考えるこ

とができる。というのは、前回のくり返しでのものの見方を文書化しておけば、もしその概念の中で経験的にうまく働かないものがあれば、それを新しい概念もしくはよりよい概念ととりかえることができる。

一般的に、このくり返しの過程ではものの見方が保全される。これによってシステム設計でのものの見方が原因となって、悪いシステムを作る可能性が減少する。また、解決の為にさらに別のシステムが必要となるような一般的な問題を解く為の別のアプローチを探し求めるというムダも省くことができる。これは定義の過程において設計の代替的な設計案を探すことだけで達成することができる。しかし多くの場合に見られるように、設計チームを代替的な設計案の開発ステップから遠ざけてしまうような心理的な環境を、プロジェクトの初期に作つてしまうことになるのは人間の本性であろうか。

ここで代替案を評価することと代替案を開発することの違いを明らかにするのは必要である。いろいろな複雑さをもったシステムにおいて、プロジェクトチームに合理的に十分よく考えられたシステムの代替案を開発させるチャンスはそんなに大きなものではない。設計のプロセスの初期に開発された考え方によって、1つの設計案がよいものだと思ってしまつたり、他の設計案が作られたとしてもそれにはらよっとした注意しかむけなくなるのが普通である。もし設計の代替案を開発することが本当に望ましいことならば、これを行うもっとも良い方法は、競争関係にある設計チームを作り、その勝者に提案したシステムの開発をまかせるようにすることである。

設計の代替案にまでたどりつく独立的なものがたくさんあれば、見方は個人的なものであまり管理されているとはいえないが、それらを評価し、もっとも適切な設計案を選ぶことは可能である。

設計の代替案はそれを開発する為の支出に値いするのであろうか。もしそ

うであったとしても、設計プロセスの各々のくり返しの中でそれが必要であろうか、その答は「イエス」であり、システムが高度のリスクをもっているとき、すなわちその実現可能性が問題となっているときには、特に必要である。設計代替案は初期の設計のくり返しにおいては大きなインパクトになりえる。システムが技術的な面で革新的であればあるほど、設計の代替案に対するその重要性は増してくる。

まとめてみると、システム設計の過程において代替案を作るように強制することは、満足すべき設計が導入されるのを保証する為の重要な方法である。

原則１０　大きく前進する最良の方法は、手頃な幅で少しづつ前へ進むことである。

〔ケース１〕

我々は組織の管理活動を実行するシステムの導入を提案する。このシステムは次に示す機能を含むことになるのであろう。

- 受注処理　（ Order Processing ）
- 伝票事務　（ Billing ）
- 受取り勘定　（ Accounts receivable ）
- 顧客配達情報　（ Customer delivevy information ）
- 工場出荷伝票の作成　（ Produce shipping orders at plants ）
- 給料記録　（ Payroll records ）
- 人事記録　（ personnel reoords ）

このシステムは、オンライン稼動をしていて、最高応答時間が５秒であるようなディスプレイ装置を活用することになろう。

このシステムの開発費用は１千万ドルで、システム導入に５年かかるであろう。

上述のようなシステムを要求しても、それがプロジェクトで、高い信頼をかもし出すことは普通あまりないようである。

〔ケース２〕

もし提案がうけいれられるならば、さらに次のように続けることになる。

我々はまずこのシステムの中の売掛処理モジュールを、期間２年間、コスト５０万ドル内で導入する計画をたてる。それによつて、プロジェクト全体の設計概念を明らかにするとともに、できるだけ正確なセケジュールとコストの推定を行なうことができる。また、たとえプロジェクトの残りが大きく変更されたり、あるいは中止されたりしても、受取り勘定モジュールはそれだけでも役立つものである。このようにアプローチすれば、確かにプロジェクトでのこのシステムに対する信頼は増すようである。

シャーマン・ブルメンソール（Sherman Blumenthal）は、彼の著書"マネジメントインフォメーションシステム"の中で、ビジネスの機能的な分野における情報システムはコンポーネント（これを彼はモジュールと呼ぶ）からなつており、これらは独立に開発できると主張している。彼はまた、コンポーネント同志の関係を明らかにして、システムの関係を調整する時間を最小にする為に、近接システム(adjacent system)を導入することを提案している。共通のインターフェースファイルをもっているときそのシステムコンポーネントは近接しているとここでは定義する。

仮説１で指摘したように、情報システムは一種の資本投資である。そしてすべての資本投資は失敗のリスクに応じて割引きされるべきである。「確かなものは死と税金だけである」と言われるのと同じように、情報システムについては、「コストはより高く、期間はより長く、効果はより少くなること

は確かである。」それゆえシステム開発をうまく進めるためには、小さな一歩よりもはるかにむづかしい大跳躍の幅をせばめることである。

情報システムを開発する理想の方針は、開発をいくつかの段階に分けて、その各々でのコストがリスクの程度に経済的に見合うように設定されていて過去の段階での経験が有効に使えるようにすることである。

図３－３にこの考え方がどのようにシステムに適用されるかが示されている。

このタイプのプロジェクトでは、最初に全体的な実行可能性が検証されるべきである。それに引き続いて、それぞれのサブプロジェクトの経済的な妥当性がテストされるべきである。

このやり方はトータルインフォメーションシステムを開発する現在流行のものとは矛盾しているが、展開的であるという利点を持つている。これはいままで積みあげてきた経験を使うもので、ほとんどの大規模システムの開発に利用されているものである。

まとめてみると、開発計画を立てるにあたっては個々のステップの長さを考慮にいれて、それをどこまで小さなステップに分割することが可能であるかを見極めることが必要である。

原則１１　疑いのあるときは文書に

これをするのには次のような良い点がある。

1. 定義の段階に続いて、それよりもさらに細かい個々の設計段階にうつていくことになる。設計のそれぞれのレベルで、それを紙を上に書くことによって、その弱点を明らかにするとともにその改善を行うことが可能になる。
2. 良い文書化は１つの設計道具といえる。というのは、アイデアを表現す

ることになると、明確でまとまった言葉を使うことになる。そうすることによって必然的に良いデザインが生まれることになる。

3. 仕事には流動性があるし、内部的に再び割り当てが変ることもあるから必ずしもプロジェクトが連続して進められるということはない。それゆえ良いアイデを書きとめて残しておかないと、それらは失われ、それぞれの設計のくり返しにおいて再び創り出されることがなくなつてしまう。

4. 十分な文書化が行なわれない限り、情報システム開発プロジェクトのあるフェーズが完成したとは思えない。

5. 開発プロセスの各段階において、品質の高い文書化がなされないと、役に立つような吟味や承認の活動は行なえないことになる。

6. プロジェクトのいろいろな段階での文書化がうまく行なわれれば、それはシステムライフを通して積みあげられ、適切なシステム保全を行うのに必要となる最終的な文書を作りあげる為のもつと低コストなアプローチになるはずである。

もっと一般的にいえば、文書ではシステムの定義に関する事項、定義の文書化に関する事項、その文書の連絡と導入に関する事項の3つが明確に区別されているがゆえにたいへん重要であるということになる。

ステップ 0.　全体的な設計概念

ステップ 1.　人事のデータベースと従業員の基本的プロファイルレポートの開発

ステップ 2.　データベースに対して例外処理と問合せ処理を行ってレポートを作成する能力の開発

ステップ 4.　人事のデータベースへのオンラインアクセスを可能にする。

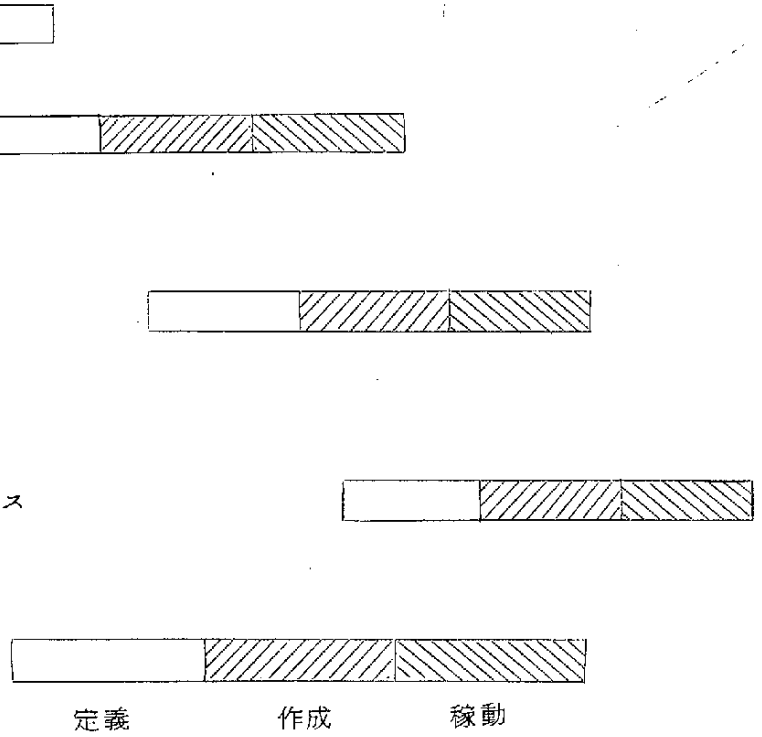


図３－３　情報システムの開発段階

原則１２　設計プロセスを構造する為に文書を使え

文書化が保守的な意味での管理の道具として使われるとは必ずしもいえない。これは設計プロセスのある部分の構造を作るのにも役立つことがある。

説明の為に、開発プロセスのある時点において、詳細なプログラム仕様書が作りあげられたものとする。ここで設計がどれだけ完成されたかを定量的に指適することは不可能である。しかしながら、これが次に示すような項目のリストは少なくとも含んでいなくてはならないと指適することはできる。

- システムのすべての分野にわたって、個々のプログラムの名前、大きさ、コーディングの構造、範囲、その使用方法が記述されていること。
- システムのすべての分野で対象とされる物理的（ physical ）かつ論理的（ logical ）レコードの定義がされていること。
- システムのすべてのプログラムについて、その機能やプログラム間でのデータの流れとともにモジュールフローチャートにおける各プログラムのクロスリファレンスが記述されていること。
- 各分野で現われるファイルのリスト
- 各分野で現われるプログラムのリスト

デザイナーに、これらの項目について文書化の要求を満たすように強いることによって、結果としては、効果的に設計の吟味をすることができるような、順序だっていてシステマティックなプログラム設計が行なわれることになる。

設計プロセスを構造化することには十分な注意をすべきである。ある線を越えるとそれは単純には受けいれられないものになる。というのはデザイナーやプログラマーは自分達の仕事を進めるにあたつてはある程度の自由を欲っしているからであり、過度に構造化ずけてしまうと設計プロセスにおけ

る創造的な要素がそこなわれる。

設計プロセスにおける構造化の程度は次の２つの要求を満足させるものでなくてはならない。

1. これが、解決しなければならない典型的な設計上の問題のほとんどを履っていること。
2. これによって、設計案は注意深い吟味ができるような形で表現され、開発サイクルの後続のフェーズで活用することができること。

原則１３　グループの仕事の進み具合をチェックし、それらの間でのバランスのとれるように。

とり組み易い問題だけを取り扱おうとするのは人間の基本的な特性である。

プログラムの専門家はプログラムの設計を好むし、システム分析者はプログラムとユーザーの間のインターフェースに関する問題を好むものである。

これらのスペシャリストをそのままにしておくと、彼らの設計したシステムは必然的に彼らの指向を反映することになり、作り出され、導入されたシステムはアンバランスなものになろう。この傾向をなくすもっともよい方法は、システム開発の構造の中に、設計問題に対して異った観点をもっているグループの間でのチェックを行い、それらの間でのバランスをとるシステムを組み入れることである。吟味をする時点において、各グループ独自の要求に関する主張が必ずしも、設計案を変更させることにはならないようにすることによって、バランスのとれた良い設計ができることを目指す。そのプロセスを経験しただれもが知っているように、これはたいへんつらいことである。けれども、関係しているすべての人の要求すべてを満足させるような、合理的な設計案を作り出す自身をもたせてくれるようなうまい方法はないのである。　開発プロセスに関連する人々の典型的なリストは次の通りである。

1. マネジメント ——— 彼らがだれであるか我々は知っている。
2. ユーザー ——— システムを望んでいてそれを導入しようとしている人達。
3. データセンター ——— システムを作り出し、情報を処理する工場のようなもの。
4. システム分析者 ——— システムを定義し、開発サイクルを活動させる人々。一般的には、プロジェクトマネジメントの機能がこのグループに付ずいしていることが多い。
5. プログラマー ——— 設計案を意味のあるコードに変換する人々
6. 手続作成者 ——— 実務部門の為のマニュアルを作り出す仕事に当る人々。
7. その他 ——— 外部にいる人々で、そのシステムに興味をもっている人。法律専門家、監査にあたる人、あるいはこれらに似た人々

これらの人々はそれぞれ異った観点をもつており、多くの組織においては、他の人々といつでも連絡をとれる立場にいることは少ない。開発サイクルのある適当な時点で、吟味と承認を行なうことで、興味をもっている個々のグループの注意をよび起して、それらの人々の参加のもとにこの吟味を続けることによって、よりよく、より信頼性のあるシステムが作り出されることになる。

いくつかの例をあげてみよう。

1. 実行手順書 (run book)を認め、データセンターのシステム管理手順を認めることによつて、これがシステム稼動がスムーズに行なわれるようなあるレベルまで書き下されることになる。
2. システム分析者によって作り出されたシステムでの要求された機能に関

する記述が、プログラマーがプログラム設計を行うのに適切であると彼らが同意するのは、開発プロセスの定義段階での設計データが適切なレベルである場合に限られる。

原則１４　計画を立てることができなければ、実行はできない

個々のシステムプロジェクトが特異な面を持っていることは、不幸なことではあるが事実である。「このプロジェクトのシステムテストをどのように行なうのか」といった質問に対して「心配するな、それはすでに終っていて、必要になったらいつでもやるさ」といった答が返ってくることがしばしばある。悲しむべきことであるが、同じ人達が次の時には異った答をするものである。

細かいところはどのように計画すべきであろうか。設計のくり返しとちょうど同じように、システムプロジェクトというものは、全体的に細かく計画するのには大規模すぎることが普通である。一般的な原則としては、その細かさの程度がどうあれ、プロジェクトマネージャーが、その全体的なプロジェクトをあるところまでレベルダウンすることは可能である。しかしながら、細部計画については彼が関係することになる次のステップでの活動の主要な集合として開発されることが重要である。このようにして、システムの定義からシステムの作成に移る時点で、プログラムや手続きの作成あるいはこれらをシステムテストへ統合化する為の細部にわたる計画が準備されるべきである。

システムプロジェクトを計画する為の理想的な方法はあるのであろうか。開発プロジェクトを計画するいくつかの工夫をのせた文献はたくさんある。これらの本では、実行されるイベントの順序を決めたり、イベントの従属関係を明らかにしたり、あるいは各イベントに配分されるべきリソースを決め

たりするのに、普通ネットワーク手法を活用している。これは様々な複雑性をもったシステムプロジェクトのいずれおいても必要なことである。

ネットワークを保全し、それを報告することはほとんどの場合、意味のあることであるが、果して大規模システムにおいてはどうであろうか。PERT, CPM あるいはいくらか変っていて、簡単な手法の中からいずれのネットワーク手法を選択するかは、相対的にみてシステムプロジェクトの成功にはさして大きな影響を与えているとは思えない。

組織の中に計画管理のシステム導入することは、強い意志と規律の訓練になる。あまり複雑な計画システムを導入したことによる失敗の結果をみても、計画システムから得られる利益に見合うような程度の組織計画システムを注意深く導入するのが得策である。

開発の次の段階までに完成されるべきタスクのリストでは、タスク間の従属性、あるいは要求されるリソース等の項目についての十分な定義がなされるべきであり、報告システムは、これらのタスクがどれ位うまく達成されたかを示すように作り出されるべきである。システム努力の範囲がそれ位であれば、精巧なネットワークを書いたり、保全したりしないでも普通はうまく行えるはずである。もしシステム努力の範囲が非常に大規模なものであれば、たぶん人手による報告システムでは追いつかなくなつてくるだろう。

プロジェクトマネジメントシステムは、システム分析者にプロジェクトに関連した様々なタクスを明確にするようにしいるときに、役に立つ道具として、最近市場に現われ始めてきたものである。これらのシステムでは、各々のシステムプロジェクトに共通している一般的なタスクの時系列的なデータをもっているから、システム分析者が新しいプロジェクトに対面したときに、そこから役に立つ知識をいろいろとり入れることができるという利点がある。

原則１５　手続きはプログラムと同じ位に重要である。

開発サイクルにおける時間とコストの関係に示した図３－２をもう一度見てみると、もし導入がだらだら行なわれると、アプリケーション開発のすべてのコストが失なわれてしまうことがわかる。導入には２つの基本的な要素がある。(1) プログラムが、本データの様々な状況のもとで信頼のできる状態を保つようにすること。と (2) システムとユーザーが気持よく働けるようにすることである。

二番目の問題がはるかに難かしいものである。この問題は、ユーザーが理解することができて、合理的である手続きを設計することによつてのみ解決することができる。これを達成する為に次のようなステップがとられることになる。

1. ユーザーの手続き書を書く人達が、ユーザーの問題をいつでも気にかけるような組織環境を作り出すように努力する。プログラムを担当している部門が、ユーザーの部門がどのように運営されているかを定義することは普通むずかしいものである。
2. プログラムの設計に見合うような手続きの仕様書を開発する。
3. 実際のプログラムと関連づけられた手続きを完成させる。
4. プログラムが導入され始められる前に、システムが手続き通りに適切に稼動するかテストを行なう。

原則１６　移行も一つのシステムである

システムの開発にあたるものは、そこに３つのシステムがあることを認識すべきである。

現在稼動中のシステム、開発されたシステム、そして現在の稼動方式から新しいシステムに移行する為のシステムの３つである。

たくさんの開発プロセスにおいて、この移行プロセスが評価されてはいるけれども、ファイルや手続きを移行するというこの過程が、開発努力の成功にとってのキーポインとなっていることまでは気がついていないことが多い。

システムの定義段階のできるだけ早い機会に、この移行プロセスそれ自体が他と異った1つの開発努力として分離され、独立されるべきである。

こうすることによって、移行システムに光があてられ、開発サイクルの成功の要因になるチェックとバランス特性に関しての検討の対象としてこの移行システムも加えられることになる。

ま と め

2節で述べられた問題の多くは、この節で作り出された原則を実際に実行することによって、解決することができる。この本の残りでは、ゴール達成の為にこのシステムプロセスに動員された人々の助けとなるいくつかの基本的なストラクチェアルスタンダードを開発することを考えることにする。

第4節　システム開発サイクルは主要ストラクチュアル・スタンダードである

概　要

プロジェクト開発の成功の基本は開発プロセスそれ自体を理解することである。なされていることとなされるべきこととの間での時間やコストの関係についての知識をもたないで、システムを開発することは大西洋を羅針盤なしで航海するのと同じ位むずかしいことである。

それぞれのステップで考慮され結論を出すべき特定の事項とともに、いくつかのステップによく整理されている開発サイクルの作業フェーズの定義は、システム・マネージャーにとって確かにプロジェクト進行の基本的な助けとなるこれが基本的なストラクチュアル・スタンダードである。このスタンダードによって「このプロジェクトサイクルのどこにいるのか」とか「成功のチャンスはどれ位なのか」とかいった質問に答えることが可能になる。

またこのスタンダードによって、前の節で議論した原則をより容易に実現化するという方法でプロジェクトライフサイクルを形成し、開発サイクルが細分化されるにつれて必要となるさらに細かいストラクチュアルスタンダードを開発することができる。

開発サイクルの記述

この本で述べられるように開発サイクルは6つのフェーズからなっている。

フェーズ	名　称
I	実行可能性の研究 (Feasibility Study)

Ⅱ　　システム仕様

（ System Specification ）

Ⅲ　　システム設計

（ System Engineering ）

Ⅳ　　プログラムと手続きの開発

（ Programing and Procedure Development ）

Ⅴ　　導　入

（ Implementation ）

Ⅵ　　稼　動

（ Operations）

開発フェーズの一般的な定義は次の通りである。（さらに細かい定義は後の節で述べることにする）

実行可能性の研究（フェーズⅠ）

このフェーズでは経済的、技術的かつ稼動的な観点からみて、そのシステムが実行可能であるかどうかという基本的な質問に答える。

システム仕様（フェーズⅡ）

このフェーズでは提案されたシステムに含まれる機能の細かい仕様書がユーザーにわかり易い言葉で書かれる。またいままでの稼動方法から新しいシステムに移行する為の移行システムの機能の細かい仕様書も作り出される。

またユーザーに約束されたシステムの作成に関する信頼性を十分に確かめることのできる技術的な設計案も作られる。

システム設計（フェーズⅢ）

このフェーズでは前のフェーズで述べられた新システムや移行システムの仕様に見合うように、ある程度細かいレベルでプログラムや手続きの設計が

なされる。

プログラムと手続きの開発（フェーズⅣ）

このフェーズでは、プログラムと手続きの関係をチェックする為の個々のプログラムテストや限られた範囲でのシステムテストを終えている手続きのマニュアルやプログラムをまとめあげた、仮のものではあるが一応のテストを終えていて文書化されたシステムが作り出される。

導　　入（フェーズⅤ）

このフェーズでは多量の本番データによってシステムの妥当性がチェックされ、それに続いて古いシステムから新しいシステムへの移行が行なわれる。

稼　　動（フェーズⅥ）

このフェーズでは稼動上の要求あるいはデータセンターの環境の要求によってシステムが修正される。またコスト的な面でのシステム稼動効率の改善が行なわれる。

システム開発サイクルと一般的な開発ステップとの関係

この本の残りでも基礎となっている上で述べた開発サイクルは、前に述べたシステム開発の3つの基本的なステップの拡張である。表4－1で示されているように、システムの定義が3つのフェーズに拡張され、システムの稼動が2つのフェーズに拡張されている。

表4－1　　システム開発

システム開発の一般的ステップ	システム開発サイクル
システムの定義 (System definition)	実行可能性の研究 システム仕様

	システム設計
システムの成就 (Systems procurement)	プログラムと手続きの開発
システムの稼動 (Systems operation)	導　入 稼　動

開発サイクルフェーズの活用

開発サイクルを定義するのに良い方法はない。ここで使われている開発サイクルは、それが理解され易いということと実際に役立つという２つの理由から採用されているにすぎない。さらには、フェーズの形で述べられている開発サイクルが多くの開発担当のマネージャーにとって有効であるという意味で正しいということが、最近になっていろいろな文献で発表されているのもこれを採用した１つの根極になっている。いくつかのフェーズの終了時点の設定がいくらかずれていることもあるけども基本的には同じである開発フェーズの呼び方は人によって様々である。けれどもこれはささいなことである。重要な点は、開発プロセスにはライフサイクルがあることとそれを管理する為の第１歩はそれを定義し理解することであるということが、認識され始めたことである。

開発フェーズへのアプローチの類似性は、私の見方と他の人達の開発サイクルの表現とを比較した表４－２を見ればまったくはっきりする。

今まで述べたことはわかっていただけたはずである。すなわち、ある組織の中での開発サイクルがここで述べられた一般的なパターンに合っていれば、またそれぞれのフェーズでなすべき仕事がその組織のすべてのメンバーによってはっきり理解されていれば、それがどのように定義されているかは重要なことではないのである。

表4－2　開発サイクルへのアプローチ　開発サイクルの部類分け

この文献で使われているシステム開発フェーズ	システム開発の他の分類					
	キャニング	ブランドン	ブリメンソール	オーリッキー	ラーデンギルダースリーブ	ガイド
実行可能性の研究	実行可能性の研究	フレリミナリィスタディ実行可能性の分析	実行可能性の評価　マネジメントの考案	研究評価	調　査	プロジェクトの開始と選択
システム仕様	細かいシステム分析と設計	システムの計画の設計	機能仕様書の完成	システム設計	システム調査 システム設計	システム分析と設計
システム設計	プログラム作成	ロジック設計	システム仕様書の準備	プログラミング	プログラミングファイル作成	プログラムの設計と作成
プログラムと手続きの開発	プログラム設計	コーディング アセンブリ テスト 文書化	プログラミングとテスト	手続きの開発 教育・移行計画 ファイルクリーンアッププログラムテスト	事務手続 システムテスト	
導　入	最終テストと移行	移行と稼動	移行と除去	移　行	並列実行	導　入
稼　動	稼動と保全	監　査	システム稼動と保全	稼　動		修正と保全

システムの定義を3つの開発フェーズに分割する

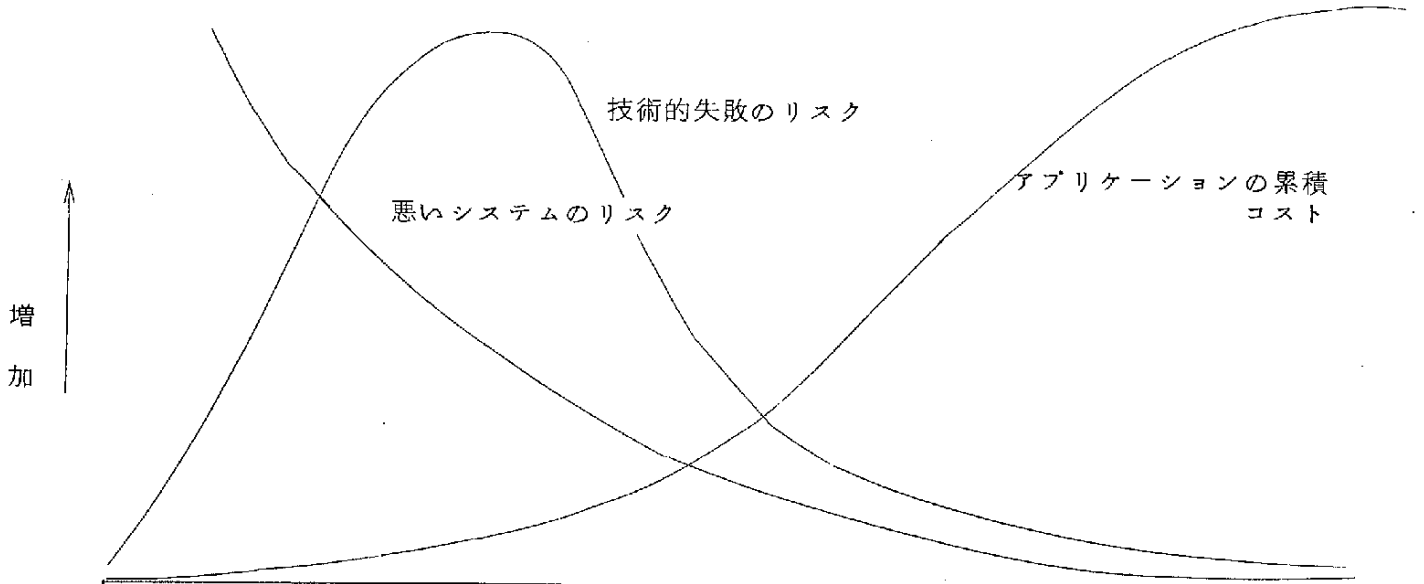
システム開発の中のシステムの定義段階を実行可能性の研究、システム仕様、システム設計の3つのフェーズに分割するのにはいろいろな理由がある。それらを以下に示す。

1. 分割することによって少なくとも3つの設計のくり返しを行ってシステムを開発することになり、それによって良いシステムを開発するチャンスを大きくすることになる。
2. 分割することによって、システム開発に興味をもっているグループがシステムを調査して、それが彼らの目的に合っていることを確かめる何回かの機会が与えられる。システム設計は最初のチェックポイント（実行可能性の研究）ではもっとも容易に中止したり修正したりできるが、最後のチェックポイント（システム設計のフェーズ）では変更は容易ではない。これらの3つのフェーズが終った時点でのシステム開発のコストは大した心配もしないでそのプロジェクトを中止できる位に低いものである。
3. システムはそれが経済的に妥当であるゆえに（あるいは思われるゆえに）導入される。多くのシステムは、その設計が非常に細かいレベルまで進まない限り、正確なコストを算定できないほど複雑なものである。設計をくり返すたびによりよいコストデータが得られる。このようにして、システム設計のフェーズで終了するシステムプロジェクトにおけるコスト見積りのまちがいの割合と同じ位にすることができる。
4. マネジメントはシステム開発の初期のフェーズにおいて大きな影響力をもっている。ある意味で、これが上の3つの文章をまとめていることになる。図の下の方に6つの開発フェーズを対応ずけて説明してある図4－1は、システム活動に影響を及ぼす要因のいくつかを表わしている。私が今

までに経験をしてきている大規模な処理システムにおいては、その開発あるいは稼動コストのクライテリアとして次のようなものを使っている。

開発コスト		稼動コスト	
フェーズⅠ	10,000ドル	フェーズⅥ	5,000,000ドル
フェーズⅡ	50,000ドル	（年当り1,000,000ドルで5年の推定）	
フェーズⅢ	80,000ドル	稼動収益	
フェーズⅢ	250,000ドル	フェーズⅥ	1,250,000ドル
フェーズⅤ	250,000ドル	（年当たり250,000ドルで5年の推定）	

定義の段階（フェーズⅠからフェーズⅢまで）のどの部分においてもプロジェクトを中止したり、遅らせたり、全く定義しなおしたりすることができる。そしてその時のコストは最大でも、140,000ドルであり、これは全開発コストの約20％、そのシステムの推定稼動コストの約36％。システムによってもたらされる稼動収益の約11％である。マネジメントの注意は特にこの3つの設計フェーズで必要であり、いったんこれらが終了してフェーズⅣへ進むことが許されれば、もはやそこでの質問は「多大な開発ならびに稼動コストを得ることができるか」というものではなくて、「これらのコストからいかに価値を生み出していくのか」という形式になることは明白である。



システムの定義			システムの作成	システムを稼動状態にする	
フェーズ			フェーズ	フェーズ	
Ⅰ	Ⅱ	Ⅲ	Ⅳ	Ⅴ	Ⅵ

図４－１　コンピューターアプリケーションのリスクとコスト

開発サイクルにおける平行開発（parallel development）の効果

平行な努力はシステムのライフサイクルを通じて行なわれる。平行な努力は次のような領域で行なわれる。

(a) インプット、アウトプットあるいはシステムのコントロールといつた一般的な領域を含めた意味で、ユーザーやデータセンターがどのようにシステムと関連するのかを記述している仕様書や手続きの開発

(b) 以前の操作方法から新システムに移行する為に必要となる仕様書や手続きやプログラムの開発

(c) システムへのデータを処理する為の仕様書とその為に必要となるプログラムの開発

(d) システムに必要とされるハードウェアーの選択と導入の検討。提案されたシステムが現在ある計算機構の変更を要求するときのみ必要となるこの努力についてはこの節の後の方で詳しくふれることにする。

それぞれの開発フェーズの終了時点において、これらの平行に進められている努力が実際に他の努力と適切に関連しているどうかを、仕様書、文書、テストプログラムあるいはそのようなもので、はっきりと調査すべきである。

このようにすれば、開発フェーズの終了点において平行して進められている活動の中ですでに終っているものが進度のおそい活動に追いつかせる為の、本来必要な遊休時間を定義することができる。チェックポイントが少なすぎるとシステムを適当に統合することができないし、逆にチェックポイントが多すぎると追いつかせる時間を費いやしすぎることになる。

システムが定義される期間にいくつかの主要なチェックポイントを設定することによって、システムについて広い見方（divergent viewpoint）をしていてシステム開発に利害関係をもっているグループが、この見方をシ

ステムの統合的な定義にも適用することについて、いくつかの合理的な確信を与えることができる。

システム開発サイクルにおけるもっとも大きなコストの要素となっているプログラムと手続きの開発フェーズを分割することはできない。というのはその多くの時間が高度に平行して進められている仕事の調整の為に費やされているからである。このフェーズ中には、それぞれのプログラムはコーディング、コンパイル、テスト、文書化という段階を踏むし、システムのそれぞれのインプットおアウトプットは手順や情報の流れの中で位置づけられる。

一般的にはこのフェーズは平行して進められた活動がお互いに完べきであるかどうかを見る実験、すなわちシステムテストを行うことによって終了する。このフェーズで平行して進められる計画の個々についていくつかのチェックポイントを設けるのが良いマネジメントの実践ということになる。

2つの主要な平行して進められる開発、すなわちプログラムと手続きの開発が利用可能なリソースの限界の中で遂行できるようにいかにリソースを分配するのが問題となる。これらの仕事を1つの統合化されたアウトプットとする為に同期化させよう試みるたびに、遅れている仕事が他に追いつく為の待ち時間が生じることになる。システムには多くのプログラムと手続きが含まれているので、このプロセスをたびたびくりかえすのはコスト高の原因となる。

処理のサイクルがまったく異っているようなシステムもある。この場合には、1つのサイクルから他のサイクルへのデータの流れはシステムの統合化がが同期化できるようなものとなっている。たとえば、日毎、週毎、月毎の処理があるようなシステムでは、日毎の処理の手続きやプログラムが単体としてテストされ、それに続いて週毎のものが、そして最後に月毎のものがテス

トできるように、このフェーズでの管理が行なわれる必要がある。このことによって、追いつかせるための時間をある合理的な範囲に制限することになるし、一方では開発フェーズ全体を通してのさらに細かい管理が可能になる。

どんな場合でも、このフェーズの最終アウトプットは、テストデータやプログラムや手続きを総合化してトータルシステムの妥当性を試してみるシステムテストである。

開発サイクルの範囲内で考えうる他のトピックス

プロジェクトの選択や設備の選択を詳しくとりあげることはこの本の範囲外のことであるが、これらのトピックスを以下で簡単にとりあげることにする。

プロジェクトの選択

開発サイクルの第1フェーズはプロジェクトの選択であると感じている著者もいる。組織に最大の利益（これはドルで換算されているか、次に高い利益率をもっているプロジェクトとの優先順位かのいずれかで評価される）をもたらすプロジェクトが、開発サイクルに組みこまれるように選ばれることがもっとも重要なことである。この主題だけでも別の本を書くのに十分すぎるくらいである。ここで我々が興味をいだくのは、潜在的に役に立つと見なされたプロジェクトをいくつか選び、それらの中から1つのシステムを作りあげていくというシステムマネージャーの機能の一側面である。

設備の選択

システム開発サイクルが設備の選択と調達のサイクルのところで混乱することがよくある。

これは、一般的には初期のシステムがそれ自身のハードウェアーを要求する為である。大規模で強力なデータ処理装置の出現によって、ほとんどのアプリケーションを含んだシステム開発の為のデータ処理の環境を作り出すことが可能になってきた。

それでそのような設備の環境条件を越えるようなアプリケーションについてだけ設備の選択過程を踏むことになる。

これは望ましい傾向である。データセンターはいろいろなアプリケーションを処理できる工場なのだから、システム開発とハードウェアーの選択の間での混乱を起こすような理由はなにもないはずである。

システムが自分の目的に見合うかどうかを検討すべきユーザーが関係のない様々の技術的な問題に関連する必要はなにもないのである。

開発サイクルお適当なフェーズで（普通はフェーズⅡであるが）新システムムが（負荷量には関係なく）データセンターの環境制約のもとで稼動することができるかどうかが決定されるということがここでの前提となっている。

例えば環境としてはバッチ処理プログラムの為に１０万バイトのコア、４台のテープ装置、１億バイトのディスク装置が利用可能であるといつた見合に表現されるであろう。ＩＢＭによって提供されているInformation Management System パッケージのようにインターラクティブでなおかつ統合化されているデータベースのもとで稼動するアプリケーションを担当するシステムデザイナーにはさらに複雑で細かい、環境の定義が必要になってくる。

システムデザイナーの興味は、これらの環境の制約のもとで要求されている機能を果しうる経済的なシステムをつくりうるかということである。もしできるとしたら、彼が必要なときに能力を提供するのはデータセンターの責任である。

しかしながら、もしそのシステム設計を実行可能にする為にさらにハードウェアーを追加する必要がありそうならば、選択と調達の過程は開発サイクルのフェーズⅡにおけるさらにもう1つの平行して進められる作業になってくる。

開発サイクルにおける文書と管理の流れ

開発サイクルの個々のフェーズでは、そのインプットとアウトプットが定義され、吟味と承認（すなわち測定し、分析し、マネジメントの決定がされること）を受ける必要がある。

図4－2はシステム開発サイクルにおける文書と管理の流れを示したものである。この図には次のことが述べられている。

1. 各フェーズでの文書やその他の技術的なアウトプットが、いかに次の開発フェーズへのインプットとなっているか。
2. 吟味と承認の機能によって、開発フェーズでの仕事の内容をもとにして、いかにプロジェクトが中止させられたり、本質的な変更を加えられたり、あるいは何の変更もしないで続行させられたりできるか。

開発サイクルでの吟味と承認

吟味と承認の機能は、各開発フェーズでのアウトプットを次のようなクライテリアのもとで調べるべきである。

システムの機能的妥当性

システムの経済的価値

完成までのスケジュール

稼動面での評価基準

だれが吟味と承認の機能を果すのになるのかは組織によって様々である。

表４－３は産業界での典型的なシステム開発機能組織を示したものであり、表４－３は組織という観点から見た場合の、吟味と承認の責任の所在を示したものである。

どんなに特定の承認サイクルであろうとも、それは公式化され、組織の中で十分に重要視される必要がある。吟味をする人によって分析されたシステムの潜在能力や落し穴の分析結果にもとずいて、乏しい資源が全体的に配分されるのであるから、その人は自分の代表する組織機能については権限をもって説明できるようにすべきである。

納得のいく吟味と承認の手順書を作ることはむずかしい。このむずかしさの原因となっているいくつかの要因をあげること。

1. システムの吟味にあたっているグループは、システムが自分達のところまで具体的化されるまでは、そのシステムの実体をつかむことはできない。
2. 吟味を担当するグループが、システムプロセスの中で積極的な役割りを努めることはないし、吟味の機能を遂行する責任を個人に割当てることもない。さらには、ほとんどの場合には、彼らがシステムに関連した実質的な責任を負うことはない。
3. 吟味が組織の非常に低いレベルで行なわれたり、非常に粗雑な方法で行なわれたりする不幸な傾向がみられる。
4. 開発グループは提案されたシステムについて保守的になるのが普通である。

5. 開発グループは、開発スケジュールに関しては伝統的にのろまであり、吟味を行うだけの十分な時間は与えることがない。

これらの問題の解決方法は2つある。1つはシステムマネジメントによる教育であり、これによって吟味の重要性を明確にすることができる。もう1つは、スケジュールは、システムで何が望まれているのかについての同意ほどには神聖なものではないことをシステムマネージャーが理解することである。（すなわち、その時に役立たないものをくばることは、その価値が疑われるということ。）

下位のストラクチュアルスタンダード

システム開発を管理する主要なストラクチュアルスタンダードはシステム開発サイクルそれ自身である。システム開発サイクルのそれぞれのフェーズにおいて、そのフェーズ中での管理に用いることのできる他のストラクチュアルスタンダードを定義することができる。これらが下位のストラクチュアルスタンダードと呼ばれる。開発フェーズごとに分類された下位のストラクチュアルスタンダードのリストが表4－4にある。それらはそれぞれのフェーズを取り扱う後節で論じることにする。ここで示されている下位のストラクチュアルスタンダードはけっして完全なものではない。これはシステムの専門家が彼らの仕事を管理する為に使っている手法のいくつかを述べたにすぎない。これらの提示が開発プロセスを作りあげ、管理するのに使えるような様々の手法の発表のきっかけになることを私は望んでいる。

ま と め

この節では基本的なストラクチュアルスタンダードであるシステム開発サイクルの概要にふれた。次節以降では開発フェーズの各々について詳しく述べることにする、そこでは特に適切に活用することのできる下位のストラクチュアルスタンダードについて述べる。

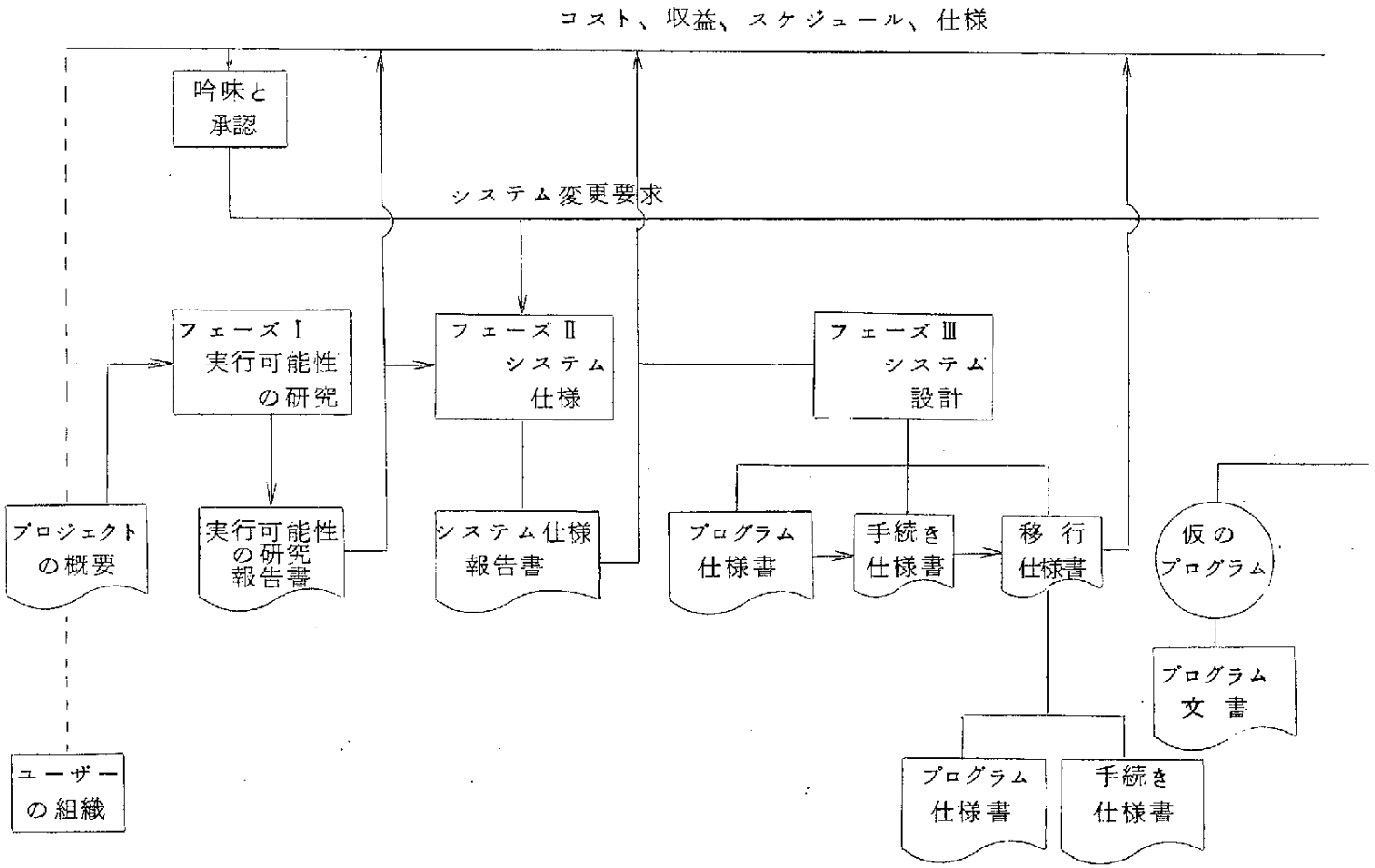


図4－2　システム開発サイクルでの文書と管理

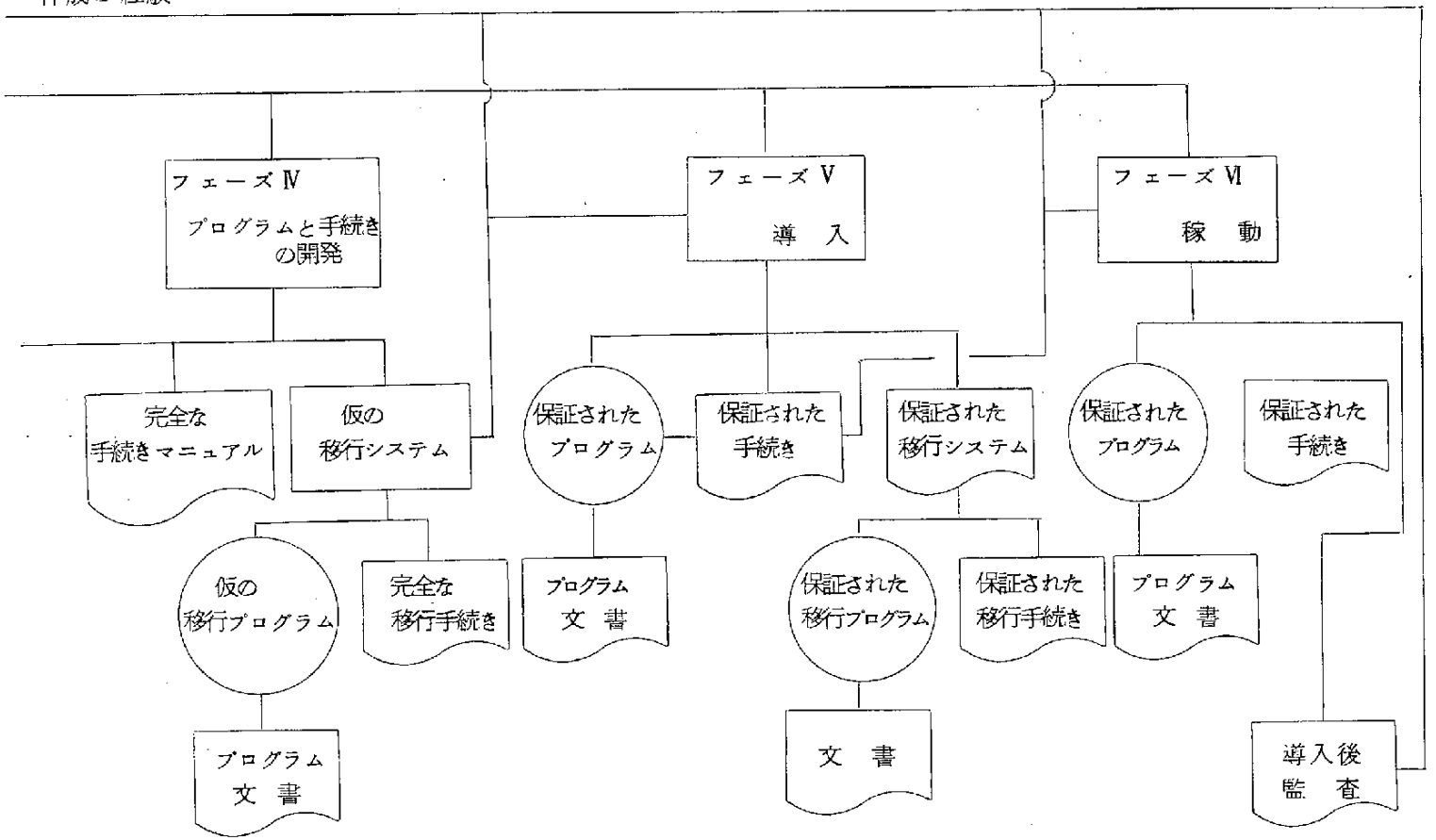

フェーズⅣ
プログラムと手続きの開発
フェーズⅤ
導　入
フェーズⅥ
稼　動
完全な手続きマニュアル
仮の移行システム
保証されたプログラム
保証された手続き
保証された移行システム
保証されたプログラム
保証された手続き
仮の移行プログラム
完全な移行手続き
プログラム文　書
保証された移行プログラム
保証された移行手続き
プログラム文　書
プログラム文　書
文　書
導入後監　査

図４－２　（続き）

表４－３　システム開発サイクル

	マネジメント			システム開発				スタッフ
	重役と経理	担当重役	データセンターのマネージャー	システム設計	手続きと管理	プログラムと設計と開発	システム改善	監査法律等
1. プロジェクトの開始要求	P・R	PCA		P. C. A			C	
2. 実行可能性の研究（フェーズⅠ）	A	PCA		P	C	C		R
A 実行可能性の研究報告		C						
a 要求の記述		P	C					
b 稼動コストの収益		C		P				
c 開発コスト		C		P				
d 開発計画								
3. システム仕様　（フェーズⅡ）								
A システム仕様の決定報告	A	C. A	C	P	C. A	C. A		C. A
a 機能的要求		P	C	C				
b 技術設計		C		P	C	C		
c 移行要求		P		C				
d 稼動コストと収益		P	C	C				
e 開発コスト		C		P	C	C		
f 開発計画		C		P	C	C		
4. システム計画　（フェーズⅢ）								
A 手順マニュアル仕様書		C. A	R	C. A	P	C		C. P. A
B プログラミング仕様書		C	R	C. A	C	P	C. A	C. A
C 移行マニュアル仕様書		P	C	C	C	C		C. A
DD プロジェクト・マネジメント	A	A		P				C. A
a 稼動コストと収益		P	C	C	C		C	
b 開発コスト		C	C	P	C	C	C	
c 開発計画		C	C	P	C	C	C	
d 移行計画		P	C	C	C			

表４－３　（続き）

	マネジメント			システム開発			スタッフ
	重役と経理重役	担当重役	データセンターのマネージャー	システム設計	手続きとプログラムの管理	システム設計と開発	改善	監査法律等
5. プログラムと手続きの開発(フェーズⅣ)							
A 手続きマニュアル作成	A	A	A	P			A
a データ・センター	C	C	C	P	C		A
b ユーザー	C	C	C	P	C		A
B プログラム作成	C	C	C．A		P	C	
C 移行手続マニュアル作成	P	C	C	C			A
D システム統合化テスト			C．A	C	P	C．A	
E システム・テスト	A	C．A	P	C	C	C．A	A
a テスト実行	C	C．A	A	C	P	C．A	A
b テスト・データの準備と評価	P		A	C	C		
F プロジェクト・マネジメント	A		P				A
a 稼動コストと収益	P	C	C	C		C	
b 開発コスト	C	C	P	C	C	C	
c 開発計画	C	C	P	C	C		
d 移行計画	P	C	C	C		C	
6. 導　入(フェーズⅤ)							
A 平列テスト	P	C．A	C．A	C	C	C．A	A
B ファイル移行	P	C	C．A	C	C	C	A
C 教育と訓練	P	C	C	C			A
D 分　割	P	C．A	A			C	A
7. 保　守(フェーズⅥ)	R						
A 導入後の評価	C．A	C	P	C	C	C	R
B システム改善	C．A	A	C．A	C	C	P	

A＝作成の責任　B＝コンサルタントとアシスタント　C＝吟味と助言　D＝承認

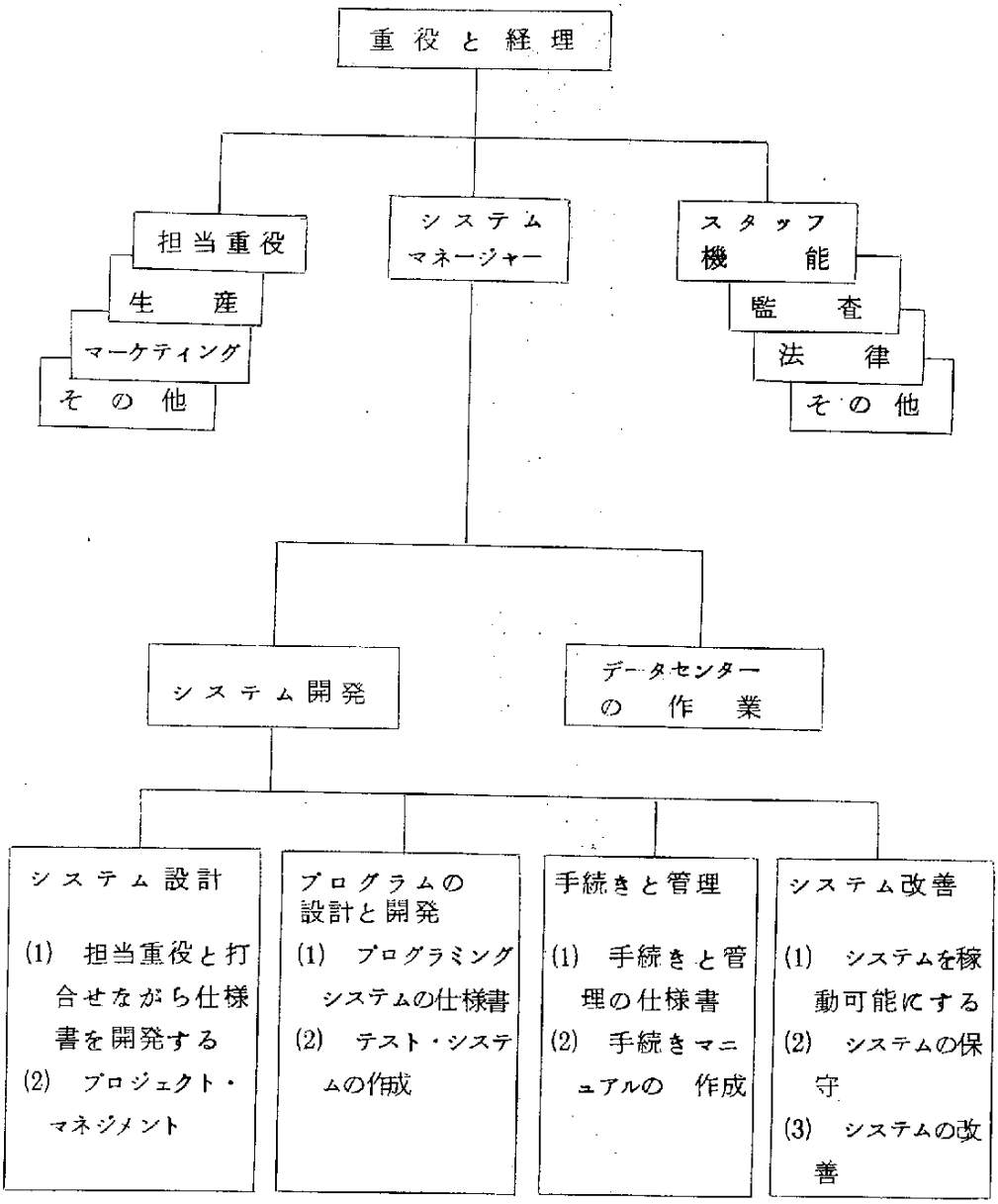


図4－3　典型的なシステム組織

（フェーズⅠ）	
実行可能性の研究	プロジェクトの概要
	コストの評価様式
	実行可能性の研究報告書
（フェーズⅡ）	
システム仕様	タスクのリスト
	質問票
	マトリックス設計(Matrix Design)手法
	フローダイアグラム
	導入計画票
	ファイル作成データソースマトクックス
	システム仕様報告書
（フェーズⅢ）	
システム設計	タスクのリスト
	システムのモジュール化
	プロトタイプシステムのシミュレーション
	仕様書
	移行日程表
（フェーズⅣ）	
プログラムと手続きの開発	タスクのリスト
	プログラム作成の慣行
	テスト作成の慣行
	手続き作成の慣行
（フェーズⅤ）	
導　　入	タスクのリスト
	除去の測定
（フェーズⅥ）	
システムの稼動	変更の管理
	システム改善

表４－４　　下位のストラクチュアル・スタンダード

第5節　実行可能性の研究フェーズ

概　要

このフェーズでは、システムが経済的、技術的、そして稼動的な面から、実行可能かどうかという基本的な質問に答える。このことは、おおまかに、図5－1でふれてはいるが、ここでは、もっと立ち入って説明することにする。

インプット

実行可能性の研究フェーズへのインプットは、それを必要とするユーザーの供述である。実行可能性の研究はユーザーの組織機能（例えば財政とか生産のような）、あるいはマジメントの吟味機能によって刺激される。どのようなタイプの要求についても、基本的に実行可能性の研究を始めるもとになるプロジェクトの概要説明書（Project outline statement）が正式に作られる。

前述のように、プロジェクトの選択に関する問題はここではふれないことにする。このこと自体は大きな論題であり、ここで書かれている問題と同様に重要なことである。ボース・キャニング（Both Canning）とブルメンソール（Blumenthal）がプロジェクト開始に関する問題について非常にくわしく論じている。

アウトプット

実行可能性の研究フェーズのアウトプットは、〝実行可能性の研究報告書〟と呼ばれるマネジメントへの報告書である。この報告書は以下のような質問に答えることになろう。

1. システムはいかなる問題を解くべきか？（例えば、部品在庫更新のため

の高い人件費、工場毎の標準化不足）この質問に対して明確な解答をするのは困難である。そうかといって不明確に答えると、結局は問題を誤って解いたり、余りにも多くの問題を解こうとして稼働コストのかかる高度に一般化されたシステムをもらすことになる。

2. システムは、これらの問題をいかに解くのであろうか？→例えば、もし工場の標準化が不足しているなら、普通は、工場での標準化を促進する手続やら報告書が書かれるべきである。

3. そのシステムは開発され得るか？この質問は実際には、それ自身をさらにむずかしいいくつかの質問に分けることになる。

   a）システムはその組織での現在の技術力の範囲に入るものなのか → 例えば、もしそのシステムが遠隔処理技術（teleprocessing tecbnology）の広範な使用を必要としているのに対して、その組織にそのような経験をもった専門家がいない場合、そのようなシステムを作り出す可能性は、はなはだ疑問である。

   b）その技術を外部の請負業者から借りてきたり、組織の中にとり入れたりすることが出来るか？

   c）このシステムのための技術は安定しているか？→例えば、システムは現在の市場にはまだ出ていない様な容量とアクセスの特性をもった大容量記憶装置が必要か、あるいはシステムが必要とするソフトウェアは信頼性があって効率がよいか。また、B／M生産システム（Bill of Material Production System）を採用している数少い会社においては、たとえ、リスト処理によってファイルを取り扱うソフトウェア（list processing file handling software）が利用可能でなくてもそれと同じようなことはできる

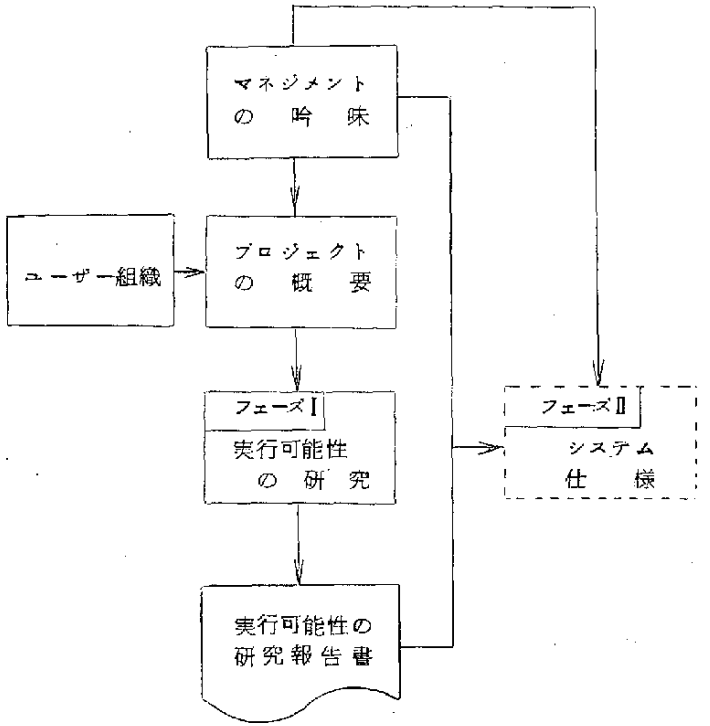
ストラクチュアルスタンダード	手　続　き　図　表	目　的
プロジェクトの概要 コスト評価様式 実行可能性研究報告書		1. システムプロジェクトを経済的、技術的、稼動的な面において行うに値するかどうかを決定するために十分詳しく調査すること。 2. システムプロジェクトの範囲を正確に定義づけること。 3. そのシステムプロジェクトを組織全体の開発計画に統合すること。 4. 設計データを、システムプロジェクトを続行していくプロジェクトグループが教育の道具として用いることができる様式に表わすこと。
		責　任
		ユーザーの分析 稼動面での実行可能性 経済面での実行可能性 システム開発の分析 技術面での実行可能性 マネジメントの評価 利用可能なリソースに対するプロジェクトの価値

図5－1　フェーズⅠ：実行可能性の研究

だろう。

4. システムは稼動的な面で受け入れられるであろうか？　この質問はしばしば省略されることがあるが、このシステムの概念が実際の作業をしている人々にとって違いものであるかどうかという事、あるいは、そのシステムのユーザーにとって、現在の方法からこのシステムで提案された稼動方法に変えることがどれほど困難なのかということには検討が加えられるべきである。もし、そのシステムを用いる人々が変化に対して抵抗を示し、システムが非常に危険を伴うようなことになれば、その原因を見きわめなければならない。そしてその原因によって、システムを中止しなければならないか、あるいはまたユーザーとシステムとの間がうまくいく様にプロジェクト開発を直して徹底的な教育上の努力をしなければならないかのいずれにきまる。もしシステムが、組織上の各部門にわたり、解かれなければならないある特定の問題を持っているならば、これも明確にされねばならない。

5. 経済的にはどうか？　システムを作成する人は、開発の見通しと、システムの稼動コストを明らかにしなければならないし、ユーザー側は、そのシステムが自分達にとって価値があるかという見通しをつけねばならない。これらは、かなり変更の可能性のある初期の推定値として認識されねばならぬが、開発サイクルの期間中に更新する機会が幾度かある。

6. 次に何が起るか？　この質問は、実際には、次の開発フェーズのための計画を述べろ、といっているようなものである。

下位のストラクチュアルスタンダード

実行可能性の研究フェーズで役に立つストラクチュアルスタンダードを以

下に詳しく述べる。

1. プロジェクトの概要

プロジェクトの概要は、実行可能性の研究の初期において、全ての関係者（来たるべきシステムプロジェクトに興味を持っている人々）が、システムプロジェクトのもたらすものに対して共通の理解を得るようにさせる為に準備された短い文書である。これは、次の3つの項目よりなる。問題の記述、問題の範囲、そして各部門の責任の3つである。問題の記述とは、何故システムが必要とされるかを簡単に書いたものである。将来の第3世代のデータセンターが現在の第2世代のシステムを効率よく稼動することが出来ないから必要なのか、あるいは、いろいろな部署に対して統一されたシステムが必要なためなのか、又はその両方なのか。問題の記述は特殊なものでなく、どちらかと言えば、一般的に書かれている（ここで述べられている問題は、調べたときに前兆としてあらわれてきていたもので、必ずしも新しいシステムによって解明されるものではないことは認識されるべきである。）

範囲の記述は、ある特定のプロジェクトの領域を述べるものである。設計者は、自由にシステムの物理的な面（基本的なインプットと手続き）を変える事が出来るのか、あるいは、設計者は現存の物理的なシステムの内容の範囲内で望まれているアウトプットを作らなくてはいけないのか。一般的に、初期の段階においてはシステム作成の範囲を限ることが望ましい。というのは開発フェーズの初めの頃には、システムの範囲を拡げる傾向があるからである。

各部門の責任の記述は、システム開発がその組織の通常の開発責任の範

囲を越えてしまうようになったとき、それがもしシステム開発の時期ならばだれが何を、またもし稼動の時期ならばだれが何を行うべきかという規則にふれている。

プロジェクトの概要は一頁にまとめるのが望ましい。プロジェクトの概要は設計仕様でもなく、ましてや実行可能性の記述ではない。それは、プロジェクト開始時に出すことのできる単なる文書であって、その内容はなぜそれを行うか、どんな制約があるのか、何を誰が行うのか、ということを述べているものである。プロジェクトの概要の例を図5－2に示す。

2. コスト評価様式

コスト評価様式（図5－3）は重要なストラクチュラルスタンダードである。もしそれがうまくデザインされていると、システム設計者に現在のシステムと計画されているシステムの両方を研究させたり、性質上稼動面で欠点にならないようにシステムに付加価値をあたえるということによって作り出すことのできる機能を考えさせたり、定義させたりする。

有効なコスト評価様式を図5－3に示す。様式の1頁目には次の様なことを書く。

1. システムの仮定稼動期間
2. システムの予想開発費用
3. 現システムの稼動コスト
4. 提案されたシステムの予想稼動コスト
5. 稼動コスト利益
6. 経済的妥当性（プロジェクト全体の価値と回収期間の見地から）

2頁目は付加価値について書く。それは、付加価値を作り出すシステム

プロジェクトの概要			
プロジェクトNo.	プロジェクトタイトル	日付	作成者

問題の記述

範囲の記述

部門責任の記述

図5－2　プロジェクトの概要

コスト評価のまとめ　　1頁

プロジェクト名　　日付　　修正名　　作成者

スケジュール

推定完成時期

A　実行可能性の研究 ＿＿＿＿
B　システム仕様 ＿＿＿＿
C　システム設計 ＿＿＿＿
D　プログラム及び手続きの開発 ＿＿＿＿
E　導　入 ＿＿＿＿
F　稼　動 ＿＿＿＿

1.　提案されたシステムの仮定稼動期間
＿＿＿＿年

2.　提案されたプロジェクトの初期投資額

	ユーザー	開発部門
A　実行可能性の研究		
B　システム仕様		
C　システム設計		
D　プログラム及び手続きの開発		
E　導　入		
F　設備の搬入		
G　上記以外の費用		
小　計		
小　計（ユーザー＋開発）		

3.　現在のシステム稼動コスト ＿＿＿／

A　ハードウェアのメインテナンス費 ＿＿＿／年
B　ハードウェアのレンタル費 ＿＿＿／年
C　ハードウェアの稼動費 ＿＿＿／年
D　メインテナンスのプログラミング費 ＿＿＿／年
E　インプットのコスト＊ ＿＿＿／年
F　アウトプットのコスト＊ ＿＿＿／年
G　ユーザーのコスト ＿＿＿／年
H　上記以外の費用 ＿＿＿／年
小計

＊通信の設備を含む

4.　提案されたプロジェクトの稼動コスト

A　ハードウェアのメインテナンス費 ＿＿＿／年
B　ハードウェアのレンタル費 ＿＿＿／年
C　ハードウェアの稼動費 ＿＿＿／年
D　メインテナンスのプログラミング費 ＿＿＿／年
E　インプットのコスト＊ ＿＿＿／年
F　アウトプットのコスト＊ ＿＿＿／年
G　ユーザーのコスト ＿＿＿／年
H　上記以外の費用 ＿＿＿／年
小計

＊通信の設備を含む

5.　稼動コストの利益 ＿＿＿

6.　妥　当　性

A　稼動コストの利益－提案されたシステムの初期投資額 ＿＿＿
B　提案されたシステムの付加価値（p2参照） ＿＿＿
C　プロジェクト全体の価値 $ ＿＿＿
D　回収期間（月）

項目2 ÷ (項目5＋項目6B) / 項目1 × 12　月

7.　承　認

承認項目	部門	サイン	日付
機　能　面	ユーザー		
手続き及び導入	部門のシステムマネージャー		
技　術　面	システム		
管　　理	開　発		
全体の目的と経済性評価	内　部		
	監　督		
	ゼネラルマネージメント		

図5－3　コスト評価様式

コ ス ト 評 価 の ま と め　　2頁

プロジェクト№　　プロジェクトタイトル　　日付　　修正№　　作成者

システムの機能 （システムの機能についての短い説明）	機能の付加価値
	$
	$
	$
	$
	$
合　計	

図５－３　（続）

の各機能とそのシステムの機能の推定付加価値の2つを記述する短い文章よりなる。

コストの妥当性様式は、各開発フェーズの間中更新されなければならない。

3. 実行可能性研究報告書

実行可能性研究報告書は、ストラクチュアルスタンダードとして用いられる。概要報告書の最初の例である。この手法は汎用性がある。スピーチにしろ、カリキュラムにしろ、雑誌の文章、あるいは本にしろ、何かものを書く時は、常に概要を書くことをすすめる。システム設計者におおまかな概要を書かせることは、概要の中で書いたフレームワークをもつ報告書を作るのに必要な項目を、システム設計者に考慮させることになる。経験のない設計者は、概要の要求するところを満足させるのは無理であろう。熟練した設計者は、多分ほとんどの要点を考慮している。経験のない設計者も熟練設計者も、彼らのアイデアを、企業に定着している方法でまとめるように強いられる。そのようにすれば、吟味の内容をまとめあげ、次の開発フェーズに必要となって追加されたスタッフに知識を伝えるのが容易になる。

実行可能性研究報告書は次のような節に分けて構成されるべきである。

Ⅰ　実行可能性の研究のまとめ

Ⅱ　システムの理論的根拠

Ⅲ　現在のシステムの説明

Ⅳ　提案されているシステムの説明

Ⅴ　移行システムの要件

Ⅵ　経済性の分析

Ⅶ　計画の要件

実行可能性研究報告書のもう少し詳しい概要は以下の通りである。

Ⅰ　実行可能性の研究のまとめ

a　解くべき問題の簡単な記述

b　行なわれる研究の性質

c　問題に対する解答となりうるシステムの簡単な記述

d　推　　奨

Ⅱ　システムの理論的根拠

a　問題の詳細な記述

b　問題の記述に対するシステムの解の説明（もし開発されるならば、その代替案についての説明）

c　コスト評価のまとめ

d　成功尺度の記述

e　システム設計における制約

Ⅲ　現在のシステムの説明（もし実行可能性の研究が新しい機能の要求に答えるものであればこの節は必要でない）

a　手続きのフローダイヤグラムとその説明

b　システムのフローダイヤグラムとその説明

c　システムのインプットリスト

d　システムのアウトプットリスト

e　システムにより維持されるファイルのリスト

f　現在の管理方針

Ⅳ　提案されているシステムの説明

a　手続きのフローダイヤグラムとその説明

b　システムのフローダイヤグラムとその説明

c　システムのインプットリスト

d　システムのアウトプットリスト

e　システムにより維持されるファイルのリスト

f　提案されている管理方針

Ⅴ　移行システムの要件

A　訓練に必要なもの

1. 新しいインプットリストとそれらの機能

2. 新しいアウトプットリストとそれらの機能

3. マネージメントに教える必要のあるもの

B　ファイルの移行

1. 新しいシステムで作られるか、あるいは移行される必要のあるファイルのリスト

Ⅵ　経済性の分析

A　コスト評価様式を説明するデータ

Ⅶ　計画の要件

A　開発サイクル全体の総合スケジュールとリソース

B　システム仕様の詳細な計画

1. プロジェクト組織の必要とするもの

2. 研究されるべき部署

3. システム仕様フェーズのタスクリスト

マネジメントの吟味

この開発フェーズでの吟味は主として次の3つの分野に重きがおかれる。

1. 提案されているシステムはマネジメントが直面する実際の問題の解となるか。
2. このプロジェクトにさらにリソースを割りあてるのに十分な経済的妥当性はあるか。
3. このプロジェクトがどのようにプロジェクト開発のマスタープランに調和適合するか。

最初の2つの点についてはここまでの議論で明白である。3番目の点についてはさらに検討する必要がある。各々の情報システム開発機能は、ある決められた期間にわたって開発されているシステムの一般的な性質を決める長期マスタープランの中に入るように決定されるべきである。各々の実行可能性の研究が進められていくにつれて、マスタープランと対比して評価されねばならないし、またその結果として、マスタープランと矛盾しないように一部変更されるか、あるいはマスタープランの変更をひきおこすかのいずれかになる。マスタープランでは、アプリケーション開発のための先行した要求やアプリケーションが開発されなければならない順序を考慮している。そして、マスタープランでは他のアプリケーションにデータを提供するための先行要求と同時に、アプリケーションのペイオフを考慮しなければならない。

例えば、販売報告システムの開発では、受渡情報の時系列ファイルをつくる受注システムが開発されることが前提となる。

第6節　システム仕様フェーズ

概　　要

このフェーズでは、ユーザーのために、彼らのなれ親しんでいる言葉で書かれたシステムの詳細な機能仕様書を作る。又、従来の稼動の方式を新しいシステムに移行するための移行システムの詳細な機能仕様書も作る。さには、ユーザーに約束したシステムを確実に作成するために、十分な詳しさを持った技術設計書をも作る。図6－1はその概要である。その内容はこの節でさらに詳しく説明されている。

インプット

この開発フェーズのインプットは、実行可能性研究報告書とマネジメントの吟味の結果正式に要望された変更とである。

アウトプット

このフェーズから出るアウトプットは、システム仕様報告書である。これはユーザーのためのシステムを包括的に書いたものであり、又詳細設計が基礎としている基礎的な技術設計についても書いてある文書である。

下位のストラクチュアルスタンダード

この開発フェーズで役に立つストラクチュアルスタンダードを以下に述べる。

1. タスクリスト

システム仕様フェーズは、大きなものであり、詳細な計画を必要とする。この開発プロセスのこのフェーズについての最も詳細な分析は、ＩＢＭの

“Study Organization Plan”という一連のマニュアルにのっている。このSOPはこのフェーズを3つの主要なサブフェーズに分類している。

1. 現在の仕事の理解
2. システムに要求されるものの決定
3. 新しいシステムの設計

もっと詳しい分類は、ガイド(Guide)のPlanning and Control of Systems and Programming Project Committee によってなされている。これでは、次のような一連のタスクを載せている。

1. 作業計画を作る
2. システムの調査
3. 新しいシステムの全般的な設計
4. ユーザーによって承認された全般的な設計
5. 新しいシステムの詳細計画
6. 移行のための全般的計画
7. システム提案書
8. 提案書の評価と承認

大規模なプロジェクトにおいては、このフェーズで出来るだけ多量の仕事を平行してするのが望ましい。この平行作業の数は主として、システムによって機械化される独立な機能の数によって制限される。このようにして、例えばマンパワーシステムを考えてみると、以下に述べるような5つの比較的独立した設計努力に分類出来る。

1. データーの取得と管理

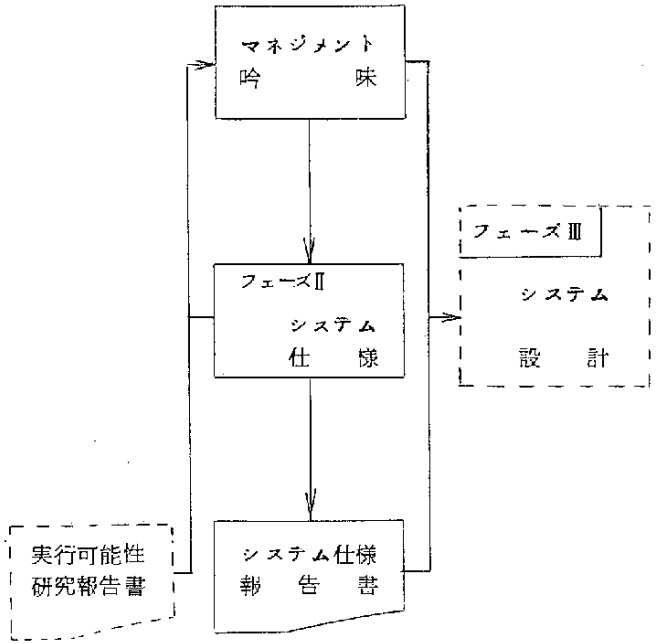
ストラクチュアルスタンダード	手　続　き　図　表	目　的
A　タスクリスト B　質　問　票 C　マトリックス設計手法 D　フローダイアグラム E　導入計画表 F　ファイル作成データソースマトリックス G　システム仕様報告書		1. ユーザーが理解できる言葉でシステムの稼動的な面で要求されるものの詳細な仕様書を作ること。 2. システムが作れることを証明するために技術的な設計を十分詳しく行なうこと。 3. プログラミングや手続きの設計を行なうことができるように、情報ベース（ infomation base ）を供給すること。 4. そのプロジェクトを続行するか否かをマネジメントが決定するための十分な費用と利益のデータを用意すること。
		責　任
		ユーザー • 稼動的な面で要求するものを詳細に述べる。 • 費用と利益について必要なものを評価する。 • 日常業務実行の問題を明確にする。 システム開発 • 詳細な技術的なシステム設計とマネジメントの仕様を作成する。 • 詳細なシステム開発スタッフの費用と新しい装置に要求される事項を列挙する。 マネジメント • 利用できるリソースに対してのプロジェクトの価値を評価する。

図６－１　フェーズⅡ：システム仕様

2. 現時点での給料処理

3. 月毎、四半期毎、年毎の給料処理

4. 従業員の利益計画

5. 人事記録

2. 質　問　票

質問票は2つの構造化された目的を達成する。

第1は、システムのための特定の質問票を作るためには、質問票を作成するところまで理解する必要があること。そしてこれによってデータ収集を行う前にある程度必要な設計研究が行なわれることになる。

第2に、多くのところからくるデータを分析に適するように、共通要素を見つけて整理する必要がある。又このデータはこみいった性質をもっているので多分熟練したシステム分析者が現場レベルの人々と一緒になって質問票を検討する必要が生じてくる。

3. マトリックス設計手法

マトリックス設計手法は、今日システムの設計を行う時にしばしば用いられる。このようなシステムの例として、ＡＰＳ、ＴＡＧがある。これらのシステムの基本原則は以下の如くである。

1. システムのアウトプット、ファイル、インプットにおいて現われるデータ要素を識別する情報要素のマトリックスが作れること。

2. マトリックスがシステム設計を出来るだけ完全に分析するための設計道具となること。

マトリックス設計手法はそれがシステムについて考える上での規律とし

てしいられるので、強力な構造化要素となる。ＴＡＧシステムは設計者がアウトプットを基準にして、全体の設計を考慮するようになっている。設計者はアウトプット（データ要素と日毎、週毎というような要求する期間により内訳された）を仕訳することにより、ＴＡＧシステムがアウトプットの要求を満たす一連の基礎的なファイルとインプットを作ることを可能にする。もし特別なアウトプット、そしていくつかのインプットとファイルが必要に応じて与えられるとすれば、システムは情報の流れから失った不完全な項目のリストを作成することになるであろう。

ＡＤＳシステムを使えば、設計者は注意深く設計された一組の様式によって、同じような操作を手でもって行なうことが出来るであろうし、各データ要素に対して別の計算規則や決定規則を付け加えることが出来る。

システムマネジャーにとって重要なことは、次のことである。

1. 一度インプットが完全なマトリックス設計に入れられると、よけいな情報を最小限度に押えた情報の流れができたという確信をそのマネージャーは持つことになる。これは設計が技術的に最適であるということではない。技術的に最適であるかどうかは、開発サイクルの次のフェーズの課題である。このフェーズの目的である設計の技術的信頼性は、マトリック設計手法によってその妥当性が確認された情報の流れによって、かなり高められる。
2. 設計はユーザー及び技術の専門家の両者が理解できる様式で記録される。

いつでも重要なことは、高度に構造化された形式で情報の流れを定義することが、設計プロセスのコンピューター化の助けとなるという事実である。ＴＡＧシステムはこの傾向を実際に示してくれている。というのは、

ＴＡＧシステムは次の５年間の内にシステム設計の方法論の全体的姿をかなり変えるであろうし、もっと複雑なシステムの問題も解けるようになってくるであろう。

4.

フローダイヤグラム（時にはシステムフローチャートと呼ばれる）は、データ（インプット、アウトプット、ファイル）とプロセス（プログラムと手続き）がシステムに必要な機能を遂行するために、結合される流れを絵で表示したものである。（図６－２）

これにより、各要素間の関係は明らかになり、書かれているプロセスを視覚で理解することができる。幾つかのプロセス表現が、設計に対する各グループの異った見方を表わすためになされる。設計フェーズで開発されたシステムの同じプロセスを表わすのにも２つのフローダイヤグラムがあり、１つはデータフローを、もう１つはコントロールフローを表わしている。（図６－３）

図表1.図６－３における図表1.は、処理が要求するものを満たす為に必要なシステムのデータフローを表わす、これは伝統的なシステムフロー図表である。この図表をもとにして明確にする必要のあるインプット、アウトプット、ファイル、プログラムそして手続きを確認せる。

図表2.図６－３における図表2.は、システムのコントロールフローを表わす、ビジネス指向型のシステムでのコントロールは、システムにおいて必要欠くべからざる部分である。それらは手続き化されていると同時にプログラム化されていて、それらの相互関係は非常に複雑である。設計プロセスの初期に設計されねばならない、ある特定の１組のプログラムと手続

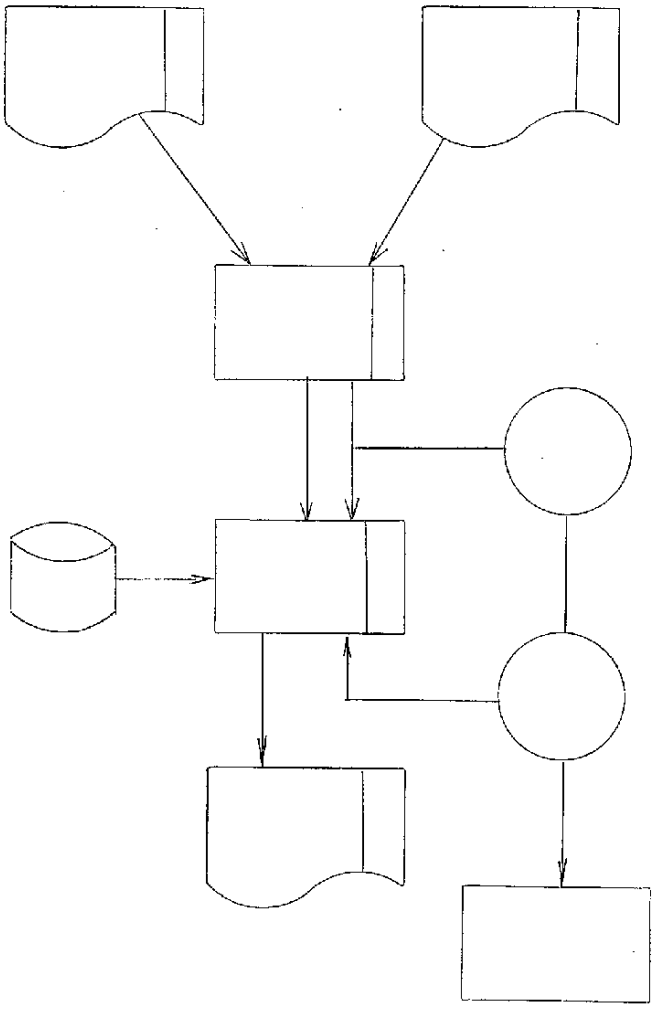
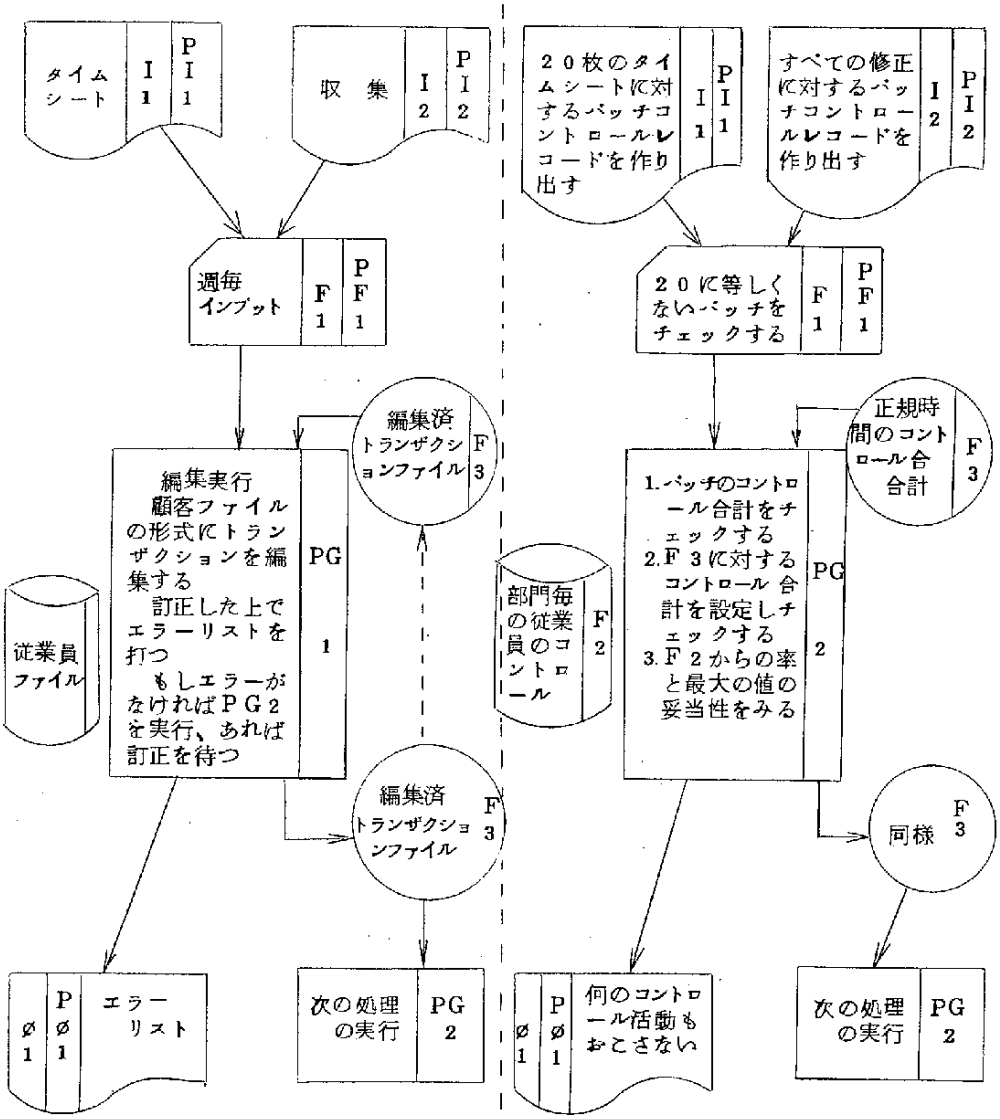


図６－２　フローダイヤグラム



略字		例
I：	インプット	
Ø：	アウトプット	PI2
F：	ファイル	はインプット2
P：	手続き	の手続き
PG：	プログラム	

図6－3　コントロールフローとデータフローを表わす図表の関係

きにそのコントロール活動を結びつけるコントロールフローダイヤグラムは、有用な構造化手法である。ここで、コントロールは十分に検討され、システムの基礎的な仕様書にまとめられ、そして、それらの吟味、注釈、修正が容易に理解出来るような様式で書きあげられる。

すべての手続き、プログラム、インプット、ファイル、アウトプットの間での関係が明らかにされ、それらの定義が仕様書の他のセクションで見つけることが出来る様に、ラベル付けされ、確認されていることが、データフローダイアグラム（図表1.）をみるとわかる。同じ様にして手続きとプログラムの間でのコントロールの関係が明らかにされ、個々のプログラムと手続きの特定のコントロール仕様書が吟味とこれに続く詳細設計のために重きをおいてとり扱われる。

5. 導入計画表

システム開発プロセスで、最も困難な仕事の1つが、移行プロセスの計画と実行である。困難になる原因として、(1)技術的システム（即ち、仕様書とプログラム）が移行プロセスと等価であると、ほとんどのシステム設計者が考えていることと　(2)開発プロセスに参加することを、システムのユーザーが嫌がることである。システムの仕様フェーズの期間では、システムがユーザーの要求するものであることを確かめることに、特に注意を払わなければならない。前の節において、議論したフローダイヤグラムにおける処理での手続きとコントロールとの明らかにされた関係は、システムのユーザーがシステムとのインターフェイスを容易に思い浮べさせるようにする為には有役である。移行計画表はユーザーとシステムの設計者が訓練、データフローの手続化、そしてファイル作成にどのくらい努力して

表６－１　導入計画表　　人事インプット様式

様式のタイトル	様式№	機　　能	困難度	必　要　な　知　識	総ユーザー数	
					部署	ユーザー
新従業員 （ＮＥ）	5. 6. 7.	新従業員のコンピューターファイルの作成	中	ダミーの社会保償番号の計算をする 新従業員に様式の説明をする１０表を作用 給与情報を求める	４０	１００
私的なプロフィルの変更 （ＣＴ）	5. 6. 7.	私的プロフィールにでてくるデーターの変更	易	カード型式は含まれていない 通常ちょっとした特殊な変更しかしないいくつかの表を使用	４０	６０
公的なプロフィール変更 （ＮＴ）	5. 6. 7.	公的プロフィールにでてくるデーターの変更	易	最も明らかなもの 通常ちょっとした特殊な変更しかしないいくつかの表を使用	４０	１００
マスターファイルの調整 （ＡＪ）	3.	上の両プロフィールにでてこないデーターの変更	難	カードがブランクであったり、適切な欄が読まれなければいけないということが大きな問題である。 表はいろいろ使う １つだけの変更というのはまれである	１	５
⋮	⋮	⋮	⋮	⋮	⋮	⋮
等	等	等	等	等	等	等

いるかを認識させる為の道見である。

導入計画表（表６－１）は、文書の型式、複雑さ、文書が用いられる部署の数、そしてそれを用いる人の数によって分類された、システムのユーザーが取り扱わねばならぬ、文書のリストを表わしている。移行計画表を準備すると、広範囲にわたる教育の手続化、訓練及び教育が考慮されねばならぬ領域がすぐさま明らかになる。『複雑なる文書の使用を教え込まれるべき８０ケ所の４００ものユーザーが実際の問題を提示することを見落すユーザーやシステムプランナーはまれである。基本的には、移行計画表は移行に関する問題への実際的なアプローチをとるように設計され構造化要素である。

6. ファイル作成データソースマトリックス

システムを計画する際に、移行プロセスの基礎的構成部分の１つであるファイル作成と移行プロセスに必要となる全てのフィールドを記載するマトリックスを開発することは、ファイル作成コストを見積もるの場合の助けとなる。次のフェーズではこれらの移行プロセスを手続き仕様書とプログラム仕様書に公式化し、そして最後に、手続きマニュアルと仮のプログラムに公式化する。ファイル作成データソースマトリックスの例と表６－２に示す。

表６－２　ファイル作成データーソースマトリックス

人事データベース

フィールド	データソース	データーソースの信頼性	移行手法
従業員名	給料マスターファイル	高	給料マスターファイルからあるフィールドを抜き出し、人事マスターファイルのインプットトランザクションを作るのにプログラムが必要か？
部署コード	部署コード表(人事マニュアル)	高	検察と転記を手作業で行なう
最終学歴	アプリケーション様式	中	検察と転記を手作業で行なう
…… 等			

7.　システム仕様報告書

システム仕様は、開発フェーズで集約されたものである。この報告書は以下の読者と、彼らの要求するものを満足させねばならない。

(a)　彼らが受け入れたり、拒否したりしているプロジェクトの全体を正しく認識しなければならないジェネラルマネージメント

(b)　稼動面でのニードに合致している形式で彼の望む機能の細かいところまでを、そのシステムが遂行していくかどうかについての判定を下さなければならないシステムのユーザー

(c) 予想される時間とコストのスケジュールの中で提示された仕様書から作業システムを作成することが出来るという確信をもっている技術設計者

(d) システム統合化においてもっとも重要な側面を自ら確かめるべき立場にある管理者と監査役

(e) 移行チーム（移行チームという概念は、移行努力をそれ自体1つのプロジェクトとみなして計画し実行する人々の必要性を強調する為にここでは用いる）

(f) 提案されたシステムのロードが現在の計算環境（conputational environment）の中で実行可能かどうか、計算環境は拡張されるべきかどうか、システムの制約は計算環境の変更を行なうのに十分かどうかなどを検討しなければならないハードウェアの獲得を担当する部門

以上のことを成すために、次のような仕様の概要が提案されている。

システム仕様報告書

（提案されている概要）

Ⅰ システムのまとめ

A そのシステムについての短かい解説

B そのシステムによって解かれる問題についての記述

C これらの問題の答となるシステムについての記述

D スケジュールのまとめ

E 成功尺度（すなわち、システムの成功の程度が測定される評価基準）についての記述

Ⅱ 新しいシステムの機能的仕様

A 用 語 集

B　システムフローチャートとその説明（システムの手続きあるいは物理的な面が強調されるレベルでの）

C　システム機能の説明

D　インプットの説明

E　アウトプットの説明

F　ファイルの説明

Ⅲ　コントロール仕様

A　コントロールフローチャートとその説明

Ⅳ　移行システムの機能的仕様

A　移行システムの大きさ

1.　導入計画表

2.　ファイル作成データソースマトリックス

3.　訓練及び教育上に関する要求

4.　組織上の責任とその割当て

B　移行システムのフローチャートとその説明

1.　システムのデータフロー

2.　システムのコントロールフロー

C　必要とされるプログラムの手続きの説明

Ⅴ　技　術　仕　様

A　システムの主要な構成部分へのブレークダウン（システムは、そのシステムの大きさによっていろいろなレベルで述べられる。例えば、受注システムについては、編集サイクル価格サイクル等のレベルがあり、またあるときには実行のレベルにおいて述べられることもある。システムのフローチャート及びその説明は、システムのモジュール化

の説明をするのに役立つ）

B　上で述べられた各々の要素については、次の様なことが必要となる。

1. 処理ロジックの説明

2. 必要なコンピュータタイムの推定

3. インプットの説明

4. アウトプットの説明

5. ファイルの説明

上の 3. 4. 5. については、次の様な情報が記載される。

a　取り扱い

b　量についてはその平均値、最大値、そして年間成長率

c　フィールドの説明では、フィールドの大きさ、コーディングの構造、フィールドのソース（インプットあるいは計算ルール）

C　これらのデータはシステムの情報フローを妥当なものにするであろうマトリックス設計の様式に記入されることが望ましい。

D　周辺の制約

1. ハードウェア

2. ソフトウェア

3. パーフォーマンス

4. 監査記録

5. 適合性からの要求（他のアプリケーションとのインターフェイス）

Ⅵ　経済性の分析

A　コスト評価様式の説明データ

Ⅶ　計画の要求するもの

A　開発サイクルのバランスを取るための全体のスケジュールとリソー

ス

B　システム設計フェーズでの詳細な計画

1. プロジェクト組織の要求事項

2. システム設計フェーズのタスクリスト

3. ネットワークダイヤグラム（もしプロジェクトがそれを必要とする程、十分大きい場合）

第7節　システム設計フェーズ

概　　要

この段階は、新システムおよび前に議論した移行システムの仕様書に適合する詳細プログラムと手続きの設計を行う。これは図7－1で図表化され、この章で詳説されている。

インプット

システム設計フェーズへのインプットは、システム仕様報告書と、マネジメントの吟味結果として必要な何らかの公式な修正書とからなる。

アウトプット

システム設計フェーズのアウトプットはプログラム仕様書、手続き仕様書、移行仕様書である。

プログラム仕様書とはコンピュータに実行される内容の細目記述書である。プログラム仕様書は、外部契約者が、仕様書からプログラムを正確にコード化出来る位の詳細なレベルで書かれるべきである。

手続き仕様書は骨格的なインプット手順の形体をとるべきであり、全てのインプットフォーム、アウトプットレポートの予備設計、手作業システムのフローダイヤグラム、各トランザクションを取り扱う手順の記述、手続きマニュアルの概要を含むべきである。

移行仕様書は、旧システムから新システムへの移行するのに必要な、プログラム仕様書と手続き仕様書である。移行作業を監督しつづける為に、プロジェクトのこの段階で、1つの活動として移行作業を設定する価値がある。

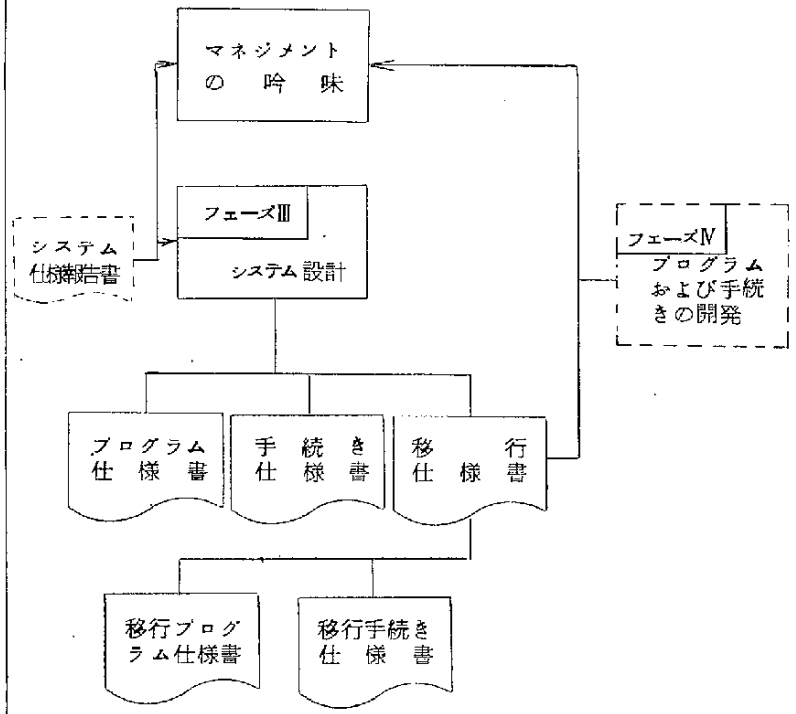
ストラクチュアルスタンダード	手　続　き　図　表	目　的
A　タスクリスト B　システムのモジュール化とインターフェースの削減 C　システムのプロトタイプ D　仕様標準 E　移行時間表		1. プログラム、手続き、移行システム領域での最終設計仕様書の作成 2. システムの存在全期間にわたつて更新され、保守しうる形式で設計仕様書を記録する。それは全プロジェクト・グループに対するシステム作業に拡張出来るだろう。 3. 開発コストと時間表および運用コストの正確な見積りを提供する、詳細な設計仕様書に基づく最終コスト分析の提供
		責　任
		システム開発 • システム・フロー、ファイル、プログラム、手続き、コントロールの詳細設計 • 修正コストの見積りの作成 ユーザー • 運用目的に対する仕様変更の評価 • 移行計画の明確化 • 財務目的に対する費用の評価 マネジメントの評価 • 使用可能資源に対するプロジェクトの価値

図7－1　フェーズⅢ：システム設計

下位のストラクチュアルスタンダード

このフェーズで適用しうる主なストラクチュアルスタンダードは次の通り。

- 適切なタスク・リストの定義
- システムのモジュール化とインターフェーズの削減
- システムのプロトタイプ
- 仕様標準
- 移行時間表

1. タスク・リスト

システム設計はかなり複雑な作業である。これを示すために、組織が当面する開発プロジェクトの規模を示す、ある典形的で、明確なパラメーターを示そう。

インプットの数	25
アウトプットの数	50
レコードタイプの数	50
フィールドの数	500
プログラムの数	100

明らかに、この規模のプロジェクトを管理しようとする際に生ずる、リソースマネジメントと意思措通の上での諸困難を解決するための注意深くて、詳細な計画が必要となる。この段階を計画する上での最初の原則は、〝分割と克服〟(divide and conguer)として最もよく表現される。すなわち、このフェーズは平行して進められるいくつかのサブプロジェクトに分割されるべきであり、サブプロジェクトは、できるだけ早い時期に個々に独立に定義されるべきである。システム仕様フェーズのタ

スク・リストの定義を議論する際、私はシステムの中に含まれる独立の機能という基礎に立って、サブプロジェクトを平行して開発してゆく可能性を述べた。ここでもう一度、このフェーズでまた平行作業を設定することになる最も論理的な基礎を提示することになるだろう。このフェーズでの作業を平行化される別の可能性は、作成される3つの異なるアウトプットに関連する。すなわちそれらはプログラム仕様書、手続き仕様書、移行仕様書である。

プログラム仕様書の作成作業

プログラム仕様書の作成作業には次のことが含まれる。

A　システム・レベルの仕様書（全てのプログラムのための共通データを含むプログラム仕様書の部分）

1. 全てのインプット・アウトプットの定義
2. システム・モジュールの定義

これはシステムを独立に作動しうるより小さいサブシステムに分割することである。各サブシステムに対し、処理機能が明確にされ、サブシステム間のデータの伝送をするファイルが決められる。

3. システム・フローチャート、ファイルの定義、テーブルの定義、高度のプログラム仕様書が各サブシステムに対し開発される。この時点で、個々のサブシステムのタイミングを必要とする記憶域の大きさを分析する。
4. システムにおいてファイルとプログラム使用によって相互参照される全てのフィールドのマスターテーブルが開発される。これはフェーズⅡで討論されるマトリックス設計手法の当然の結果となっている。
5. 先ず概念的なテスト・プランを開発する。次にシステム・テストの

方針を明らかにし、ついで個々のサブシステムをテストする方針を開発すべきである。

B　プログラム・レベルの仕様書

1. 高レベルの仕様書を吟味する
2. プログラムのための詳細な文書による仕様書の作成
3. プログラムのためのフローチャート
4. プログラム・テスト・プランの作成

次のような順序で作業部隊が徐々に拡大するのはあきらかである。

1. A(1)と(2)はシステムのマスターデザイナーにより行なわれる。
2. A(3)から(5)までは個々のサブシステムに責任をもつデザイナーにより行なわれる。これらの作業はマスターデザイナーの監督を受ける。
3. B(1)から(4)までは、プログラム、デザイナーにより行なわれ、サブシステム・デザイナーの監督を受ける。

システムの設計は、構築が進むにつれてシステムおよびプログラムの建造物（architecture）として、多くのプログラマーにより書き上げられてきた。これはこの本の最初の方でその類似性を検討した製品開発と非常によく対応した一般的な見解である。

手続き仕様書の作成作業

このフェーズでの手続き仕様書は、インプット・アウトプットが何から構成され、この情報がコンピュータールームから物理的にいかに流入・流出するかを詳細に確定する必要性を意味している。またシステム・トランザクションを文書化しはじめることが必要であり、そうすればこのフェーズの終了時点で、ユーザーの部門がこの新システムの下でいかに機能するか目で見ることが出来るだろう。

これを達成するために開発タスクの次のようなリストが考慮されねばならない。

A　インプットフォームの設計

1. どんな情報が必要であり、それはどんなフォームであるべきかを確定するために、コンピューターシステムのフロー・ダイヤグラムとインプットの定義を使う。

2. ユーザーと協力して、最適なフォーム・デザインを確定する。

B　アウトプット・レポートのレイアウト

1. 各々のレポートにどんな情報が書かれ、どんな順序でかかれるべきかを確定するために、コンピューターシステムのフローダイアグラムとアウトプットの定義を使う。

2. ユーザーと協力して、各レポートがいかに使われるかを確定する。

3. 各レポートのレイアウトを設計する。

C　手作業の外部システム・フロー・ダイアグラムを描く。

1. コンピューターシステムのフロー・ダイアグラムとプログラム仕様書を使って、トランザクションがコンピューターシステムを通じていかに流れエラー訂正がいかになされる。あるいはまた、システムは何を管理しようとしているかを確定する。

2. 次のことを確定するためにユーザーと協力する。

   a　どんな要員が使用可能か

   b　彼の部門が供給するものは準備出来ているか

   c　彼が計画した組織はどうか

   d　新しい仕事の構成はどうか

3. 伝票作業の流れの概要

4. コントロール手続きの概要

5. エラー訂正手続きの概要

6. 手作業システムの作業流れ図の作成

D トランザクション・マトリックスの準備

1. プログラム仕様書を使って、各トランザクションが、そのシステムでいかに処理され、全てのトランザクションがいかに相互作用するかを確定する。

2. トランザクションの相互作用を示すマトリックスを描く。

3. 各トランザクションがシステムでいかに処理されるかについての概要を作成する。

E ユーザーマニュアルの草案作成

1. 次の点についてユーザーとプロジェクト・リーダーが協働する

a ユーザーマニュアルに必要な項目を決定する。

b どのような種類の要員がマニュアルまたはマニュアルの一部を使用するかを決定する。

c ユーザーマニュアルをどのような様式にするか決定する。

移行仕様書の作成作業

移行過程は2つの主要な面を持つ

旧システムのデータを新システムが処理しうる形式に交換する手続きと、プログラム。

新システムを受け入れ、適切に利用するために、ユーザー側の条件を整えるのに必要な教育。このことを達成するために、次のようなタスクを完了しなければならない。

A ファイルの移行

フェーズⅡで、われわれは達成されるべきファイル移行プロセスを一般的に明らかにしたファイル移行テーブルを作成した。このフェーズでわれわれはこれらのタスクの詳細な仕様書を作成すべきである。

1. 各々のタスクに対し、次の項目からなる移行仕様書を作成する。

   a　データ・フロー図

   b　手続き仕様書

   c　プログラム仕様書（必要な場合）

B　導入教育

1. 移行されるいろいろな部署で実行されることになる。教育と移行作業のスケジュールを作成する。

2. 各タスクのための仕様の作成

このフェーズはかなり長期になりうるし、プログラム、手続き、移行計画等のグループが関連した活動を含むことになるので、この段階でのきめ細かいスケジュールを設定する為の注意深い計画表を作らなければならない。各グループ内での吟味、品質管理、あるいは諸仕様書の内容がうまく一致することを確かめるために協同して行なわれる吟味などに適切な時間をさかねばならない。フェーズⅡで討論したように、チェックポイントの数は費やされる資金と強い関係をもつ。

2. システムのモジュール化とインターフェースの削減

デジタル装置の設計は共通の構成要素の数を最小にするような論理機能をまとめあげるとか、これら構成要素間のインターフェースを削減するとかいった、いくつかの問題に関連している。これらのテクニックをうまく適用した結果が、拡張可能性と保守の容易性をもったモジュールという美

学的にも好ましい設計である。ディジタル装置の設計者がより単純な設計を探索するときに、彼の助けとなる。性質上数学的な、多くの強力な設計手段をもっている。これらの設計手段は、設計者ができるだけ設計を正確にしようとすることと、論理条件を数学的に記述する能力をもっているという2つの理由から役に立つことになる。

情報システムの設計者は同じ目標を達成しようとしているのだが、まだ問題を正確に記述する方法を開発していないし、数学的に適切に表現しうる情報システムの構成要素はまだありえない。だからシステム設計者は、処理ロジックの要素化とシステムのインターフェースの最少化を創造的にものごとを組み合わせていくことによって果たさなければならない。

高度に創造的である設計合成の過程自体のうまい表現法は今後の開発を行っているような状況であるけれども、設計合成の過程の結果を明瞭にすることはできる。果たさねばならないことは、システム設計の全てのレベルのモジュール化である。モジュール化にはいくつかの特徴がある。

(A) 機能のユニークさ。　各モジュールはそれがシステム設計のどんなレベルであっても、システムのためのユニークな一連の機能を果たすべきである。このようにすれば、もしこれらの諸機能を持っているシステムに修正が必要となった時には、特定のモジュールだけを対象にすればよいことになる。

(B) 論理的な完全さ。　モジュールはその仕様にモジュールへのインプット・データとモジュールからのアウトプット・データおよびデータ変換を実施するための処理規則が書かれている場合に限って、論理的に完全な設計機能となる。モジュールが論理的に完全な場合、各モジュールは詳細に設計され、コード化され、システムの他の全てのモジュールとは

独立にテストすることができる。だからモジュール化は作業を行うに際して、平行作業を可能とし、大規模プロジェクトが実施される時にはいつでも基本となるものである。

(C) モジュール間のデータインタフェースの最小化。 辞書によると、インタフェースとは2つの物体間又は空間間の共通の境界と定義してある。情報システムの場合には、データ・ファイルがそれを必要とする計算過程間でのインタフェースである。

ある意味において、構造の応力図形と情報システムのモジュール化と間には類似性がある。構造の各接合点は接合点の構造エレメントの張力でバランスを保っている。ある構造エレメントが変化すれば構造組織の数ケ所の接合点で張力に変化が起る。構造エレメントが情報システムの処理モジュールであると考えた場合も同じ状態を見ることが出来る。この場合の接合点は処理モジュールであり、バランスを保たれているデータ・インタフェースである。システムの設計中に処理モジュールの要求事項は変るかも知れない。この変化は、順々にシステム化のインタフェース点での処理モジュールの要求事項に変化を引き起す。

上記の類似性からモジュール間のデータ・インタフェースの数を最少にするよう努力すべきだということが解る。明らかにインターフェースの数が少なければ少ないほど、処理モジュールはそれだけ独立性を保つことができ、設計要求の変更からもたらされるショックの大きさはそれだけ小さくて済む。

(D) 包含性（Comprehensibility）。 システムをモジュール化する第4のそして多分最も重要な理由は、全体的な見方をしないと、単純にはこれを理解出来ないということである。モジュール設計は、システム

を細かさの異ったいくつかのレベルに分割することを可能にする。この分割では順々に各モジュール・レベルが仕様書に書かれているモジュールより更に細かくされていく。大規模なシステムには必然的に性質上モジューであるといえる。そうでなければ決して組み上げることはできないだろう。そこでこの特性を必然的なものと認め、モジュールを積極的に利用することが助言される。

(E) モジュールのレベル。 モジュールとサブルーティンとを同義語だとみなすことがよくある。実際にはサブルーティンはシステム設計者が関係するモジュールのより下位の有用なレベルのことである。図7－2はこれらのモジュールのレベルを示している。それは又、仕様書作成過程の階層的性質を示している。フェーズⅡでの仕様書はプログラム・レベル程度までの設計に対して作成すべきである。フェーズⅢでの設計者はフェーズⅡで定義された、最低のレベルの仕様書を変更できる自由度を持つべきである。同様にフェーズⅢでの仕様書は、このレベルでの修正はプログラミングフェーズ（フェーズⅢ）で起こるであろうという認識のもとで、サブルーティン・レベルまで含むべきである。

実際仕様書とは、最も総体的なものから、モジュールへのインプットモジュールからのアウトプット、次の処理で使用するため保持しているデータといった実施される処理機能の最も低位の細目レベルまでのシステムデザインの各レベルでのモジュールを記述したものである。各モジュールは次のような情報を含んだ形で定義されることになる。

A モジュールの記述

1. 目　　的

2. インプット

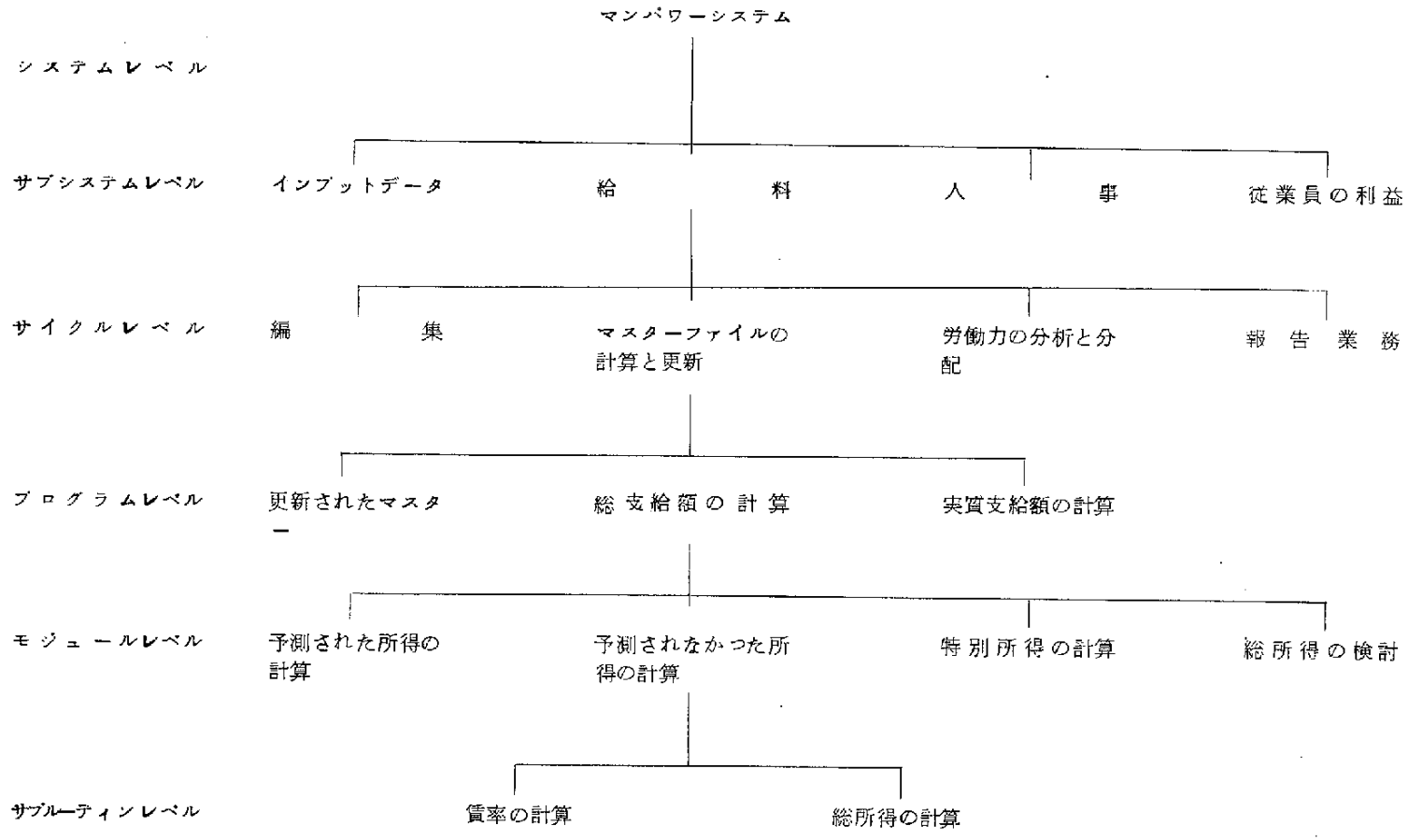


図7－2　システムモジュールのレベル表示

3. アウトプット

4. 保持しておくファイル（又はデータ・エレメント）

5. 果たすべき処理

6. 次のレベルのモジュールについての簡単な説明

仕様書は設計過程で次々に作り上げられる。そして仕様書が作り上げられる速度が設計作業の進捗をよく反映している。

3. システムのプロトタイプ

開発過程は危険度が高い。製品開発に関係している人々は、危険を少なくする最もうまい方法の一つは開発される製品のプロトタイプを作ることだということを学んできている。プロトタイプは、一般に開発サイクルが継続されるにつれて、精度と複雑さのレベルが増大してきた時点で作られる。

情報システム開発にたずさわる人々は、プロトタイプシステム開発のコストが高く、プロトタイプの性質についてある混乱があるため、普通は製品開発者の先導に従がわなかった。

ちゃんとした定義はいくつかある。第1に、プロトタイプはモデルであるから、プロトタイプは、それが説明しようとしているもののある特定の特性だけを表わす。かくしてモデルは飛行機の実物模型になりうるし、それは飛行機を目で見ながら学ぶのに役だつ。それはまた飛行機の数学モデルにもなりうるし、これは気体力学の性質を学ぶのに役立つ。更にそれは工場生産に移される前にきびしくテストされる設計段階の最初の飛行機でもある。

プロトタイプはさまざまな目的に役立つ。そこで、情報システムのプロ

トタイプを設計する場合、プロトタイプの目的が何であるか決める必要がある。プロトタイプを作成するに当って次の様ないくつかの理由がある。

(A) プロトタイプは重要な基本水準標（ベンチマーク）である。実際に役立つプロトタイプを作るには多くの設計上、経済上、製品上の問題が完全に解決されねばならない。システム・プロジェクトプロトタイプは、システム設計フェーズの間に、シミュレーションによって達成することができる。シミュレーション技法を選択する際にいくつかの決定段階がある。複雑なタイミングと待ち行列の関係（航空座席予約がその典型であるような高い需要度をもつオンライン・システム）を含む設計の重要な面は、正確な言葉で、アプリケーションロジックやシステムの構成要素をモデル化することで最もよく表わされる。高度の解析シミュレーションと実行する。汎用シミュレーション言語にはＧＰＳＳや、ＳＩＭ－ＳＣＲＩＰＴがある。コンピューターのＩＢＭシステム３６０系統のアプリケーション・システムをシミュレートするのに役立ち、パラメータ化したハードウェア構成要素のビルディング・ブロックを利用することのできる特定のシミュレーション言語はＣＳＳである。これらのシステムでモデル化される適用分野は、極端に詳細な仕様書を必要とするが、性能評価に関する程度の高い解答を次々とアナリストに与える。

一般的にいって、主にバッチ処理を指向している情報システムは、１９６０年代のタイミング・プログラムを発展させたものであるモデル言語で、うまく表現することができる。

これらのうち最も広く利用されているものは、ＳＣＥＲＴというソフトウェア・パッケージである。ＳＣＥＲＴでは、ファイルやレコードの性質、ファイルの活動（ファイルでのレコードの活動しないものに対す

る活動するものの割合）、処理機能、およびハードウェアとソフトウェアの構成等を表わすパラメータによって、プログラムまたはプログラム体系を記述するように考慮してある。アウトプットは、提案中のシステムのハードウェアの構成要素ごとの使用状況を表わす一連のレポートである。アウトプットレポートはどこを変更したら提案中のシステムの性能を改善することができるかを示すのに使うことができる。

要約するとシステムの設計フェーズの結果として情報システムの設計案をシミュレートすることは貴重な基本水準標（ベンチマーク）を提供する。シミュレーションを実行するためには、設計をかなり細かいレベルで行う必要がある。このように、シミュレーションのアウトプットは適切なポイントを充分考慮した設計であるという証拠であり、そしてまた設計をいかに改善するかという性能に関するデータを提供するものである。

(B) プロトタイプの特徴は目で見ることが出来ることである。システム・デザイナーが直面するより困難な問題の1つは、システムを設計してもらう側の人々が、稼動システムが実際どうなるのか理解することが出来ない（もしくは理解しようとしない）ことである。新製品の開発サイクルのこの段階では、実物模型が作られる。それは触れることが出来るし、においをかいだり、テストしたりすることもできるし、将来参加が予定されているデザイナーやユーザーが多くの様々な方法で分析してみることが出来る。プロトタイプ情報システムを開発するのは非常に高くつくため、これまでこの分野ではプロトタイプが殆んど作られなかった。例外としていくつかの大規模な命令・統制システムがあるが、この場合は軍が、利用者と一緒になって、相互作用をもつ構成要素のプロトタイプ

をテストするのにかなりの額の投資をしてきた。経済的に見て、状況は一般的に開発サイクル内で妥当とみなされた費用で、プロトタイプシステムを開発し、テスト出来るように変りつつある。MARKIV、COGENT2とかGISとかいったタイプのファイルマネジメント言語を使ってプログラムを作成する費用は、小グループのデザイナーとかユーザーがテストしたり評価したりしたいと思っているデータ・ベースとか、インプット手続きとか、アウトプット・レポートの簡単なテストプログラムを手続き指向の言語で作成する費用よりずっと少ない。これはインターラクティブなタイムシエアリングのもとでこれらのシステムが開発され、ユーザーテストが行なわれる場合には特にそうである。プロジェクトの規模が増大するにつれて、運用上役に立たないと考えられているプロトタイプシステムの開発が、システム開発プロセス内で受け入れられるようになり、望ましいものとなるだろう。プロトタイプシステムの設計は非常にむずかしい問題である。もしプロトタイプがあまり単純である場合には、役に立たないだろうし、もし全てを包含しようとすると、トータルシステムのプロトタイプを作る場合の費用でしか作れないだろう。

4. 仕 様 書

4種類の仕様書がこの段階で作成される。

A　プログラム仕様書

B　手続き仕様書

C　移行仕様書

D　コントロール仕様書

プログラムおよび手続きの両仕様書を開発するにあたっては、修正を管理し、関係先へ連絡するための計画に充分注意を払うべきである。ＩＢＭシステム３６０のＤＯＳのプログラム仕様書はテキストで２万頁以上にわたって記載されている。かなり大規模な商業アプリケーションのプログラム仕様書は数千頁のテキストに相当するものになるだろう。このように大規模なテキストの内容の保守は、有効にコンピュータに委ねることが出来るための１つのファイル保守作業になる。もしこの作業が行なわれれば、副次的な利益が得られる。というのは主要な設計情報をシステムに組み入れるインプット文書が設計されることになるからである。これらのうちいくつかの例は次の通りである。

1. 図７－３はインプット・アウトプットおよび各アウトプットの行き先を示したシステム内の各処理を記載したものである。それは典型的なデータ、またはシステムフロー・ダイヤグラムと内容上同一のものであるが、容易に保守することが出来るし、各処理が相互参照をするファイルを自動的に見つけ出すことができる。
2. 図７－４はファイルの物理的性質を記載したものである。レコードの特徴は原始言語（ｓｏｕｒｃｅ　ｌａｎｇａｇｅ）を活用して設計の段階で定義することが出来る。原始主語の形式をよく注意して構成すると、各フィールドに存在するレコードのタイプとファイルを表わす。相互参照リストをプログラムで作成することができるだろう。
3. 図７－５はシステム内の各フィールドを記載したものである。『システム・モジュールとか、処理仕様書といったその他のプログラム仕様書は、物語のような形式であり、この節の方でモジュールについて議論した際に指摘したように、標準的な仕様書の概要を作成することによって、

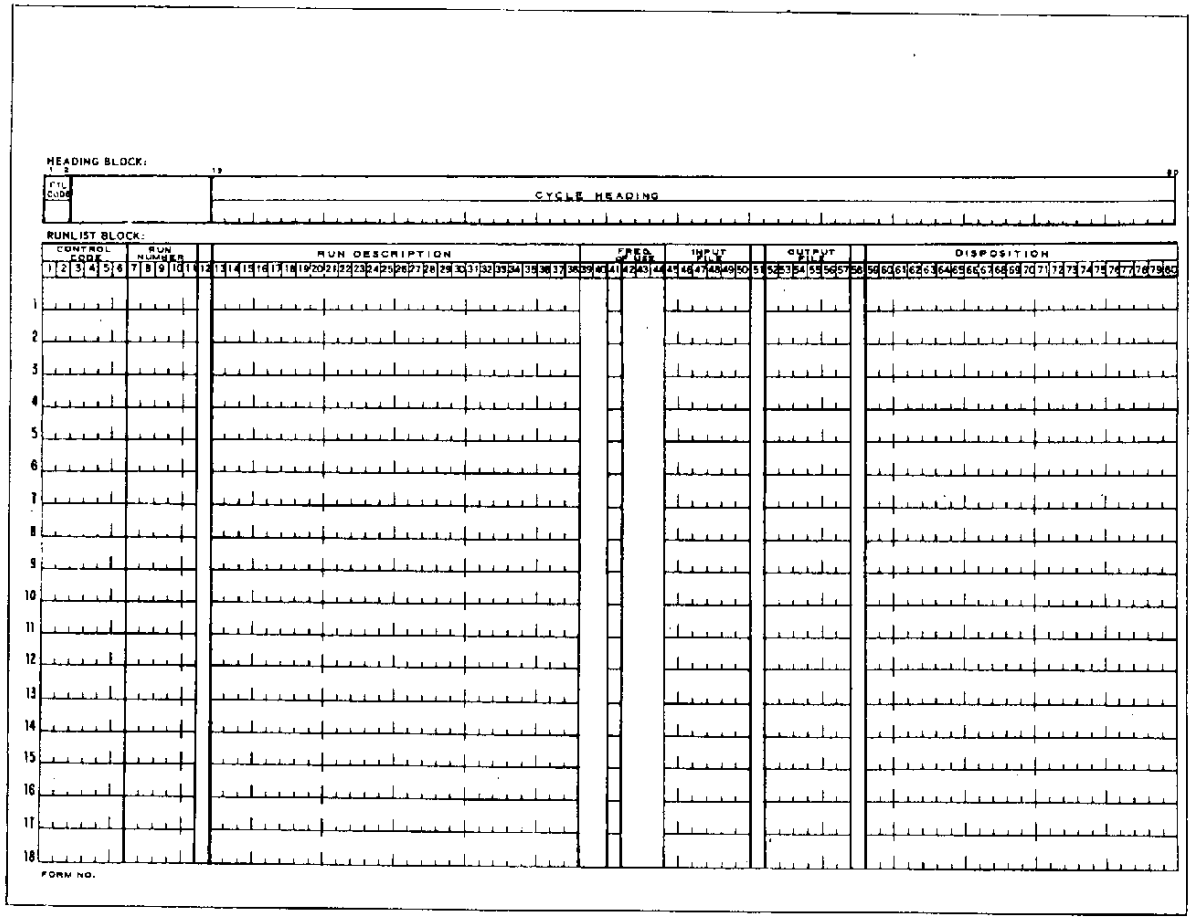


図７－３　ランリストのフォーマットシート

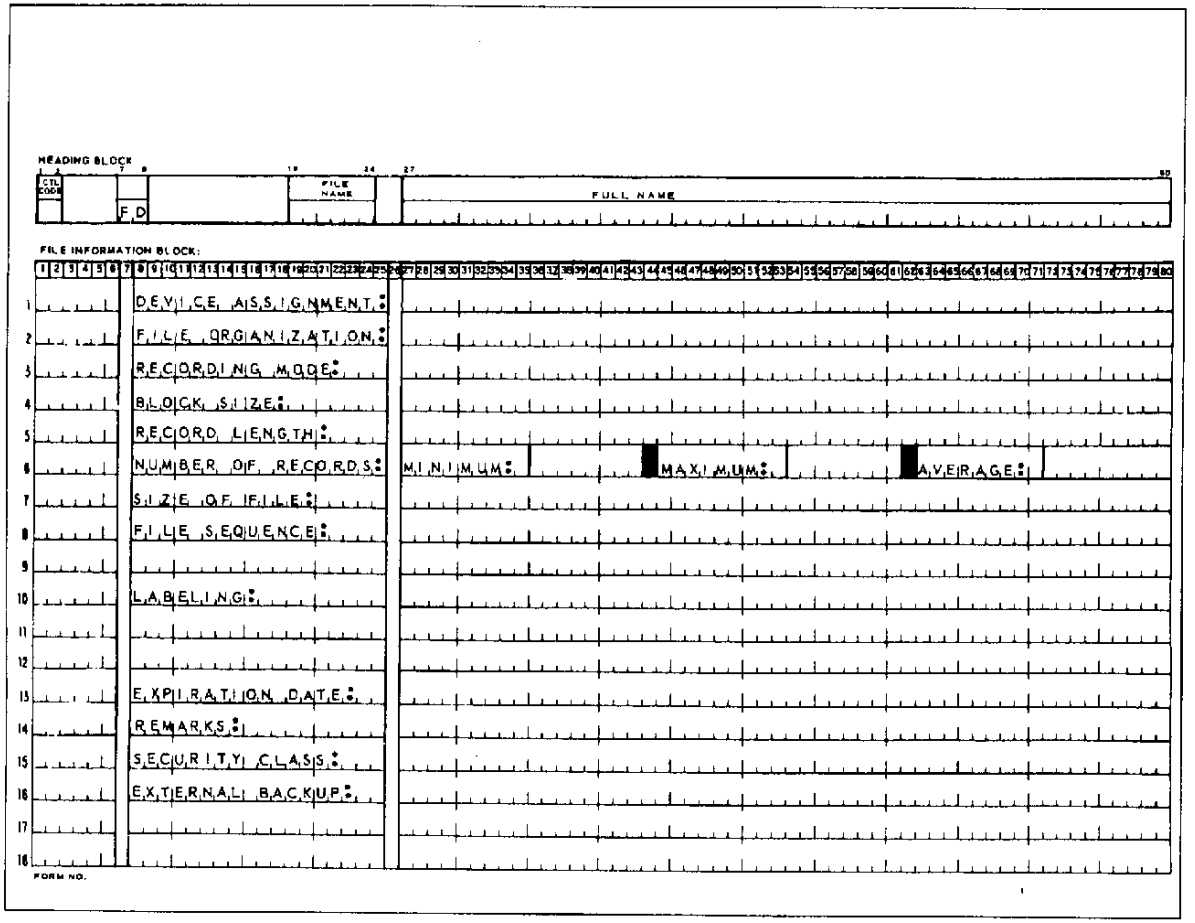


図７－４　ファイルのフォーマットシート

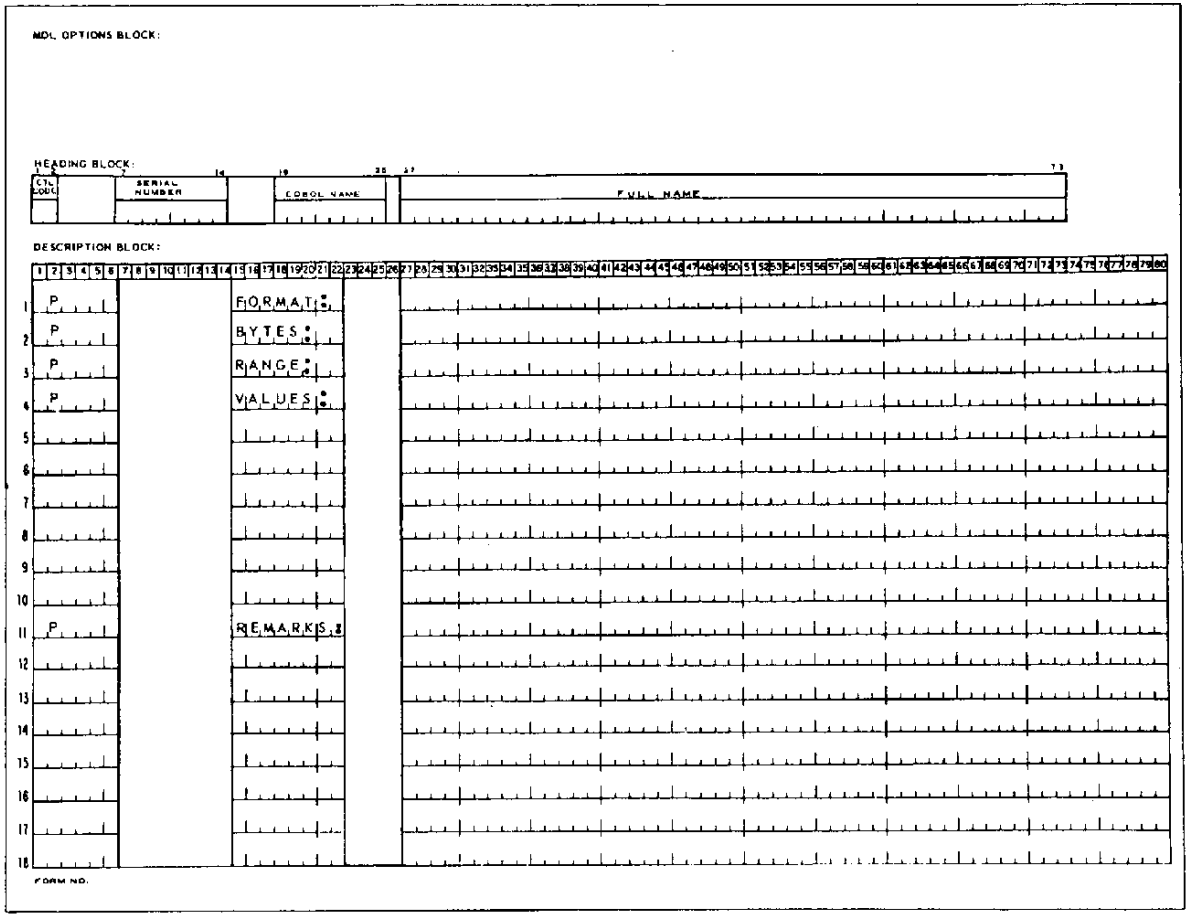


図７－５　フィールドリストのフォーマットシート

もっともよく構成することができる。』

実際のところ、コントロール仕様書の内容は、プログラムおよび手続き仕様書に記載される。しかしながら、必要とされるものは、コントロールをシステムの全てのインターフェースに関連づけ、どんなコントロールが各インタフェースに関して設定されるのかを記述し、このコントロールがシステムを通じて行なわれる方法を記述した全般的な仕様書である。フェーズⅡで説明された平行作業を管理する為の流れ図とデータ流れ図が、この設計フェーズでの適切な細目のレベルを示すのに拡張されるのが望ましい。

移行仕様書は、プログラムおよび手続き仕様が一緒になったものであり、これらを開発するためには次の規則を守るべきである。

5. 移行時間表

テレビで月着陸を見守るわれわれには役に立たないが、航空宇宙士が作ったシナリオや脚本を使っていたということには驚ろかされた。それらには航空宇宙士が多分出くわすものや、一定の時系列の特定の時点で、多分行うべきところのものが記載されていた。『移行を行う前に実施されねばならない出来事を記述したシナリオを準備することは、移行過程を計画する手法としてたいへん役立つ。』移行過程の複雑さを説明するために、1つの大規模なシステム（それに私は詳しい）を考えよう。移行されるべき各場所でのリードタイムは16週間必要であって、その期間中に40以上の実施すべき行為があった。

このような複雑な仕事を定義する作業は簡単な仕事でないばかりか、訓練、教材や手続き書を作成したり、移行計画を実施するプログラムを作る

ことも簡単ではない。有効なストラクチュアルスタンダードは移行時間表や脚本文書にして、システムのユーザーと開発担当者の両者がその文書をもとにして吟味したり、議論できるようにすることである。もし計画がうまく提示されるならば、その計画は開発に反対の側からも、その計画に必要な修正が引きだされるだろう。

移行時間表の例を図７－６に掲げておく。

システムの名前	年月日
オーダーエントリーシステム	6／5／69
活　動	さし向けられる要員
１６週間前	
システム概念の吟味する	
担当マネジメントとともに	
限られた責任の中での	
人的資源の必要量をきめる	
移行対象部門で	
導入のスタッフを選ぶ	
労働組合の官吏に連絡する	
１４週間前	
マスターファイルを作る方針について吟味する	
マスターファイルを作る手順（用紙、ラベル、作業票）	
について吟味する	
１週間前	

図７－６　移　行　時　間　表

第8節　プログラムおよび手続きの開発

概　　要

このフェーズでは、手続きマニュアルとプログラムを完備した、仮のものではあるがテストがすんでいて文書化されたシステムを作成する。システムは、個々のプログラムテストおよびプログラムと手続き間の関連を確認する為の限定されたシステム・テストを終了している。その状況が図8－1に概略的に記載されているし、この章の残った部分でさらに詳説されている。

インプット

このフェーズへのインプットは前のフェーズで作成された、プログラム、手続き、移行仕様書およびマネジメントの吟味の結果として公式に要求される変更項目とからなる。

アウトプット

このフェーズのアウトプットは前のフェーズで詳しく述べられ、設計されたプログラムと手続きである。このように、1から3までのフェーズは仕様設計の段階であるけれども、フェーズ4は作成の段階である。その製品は仮であるが文書化されたプログラムと手続きを集めたものである。ある一定量のデータによるアウトプット製品のテストは、そのシステムが統一体として実行しうるということを証明しようとするものであった。

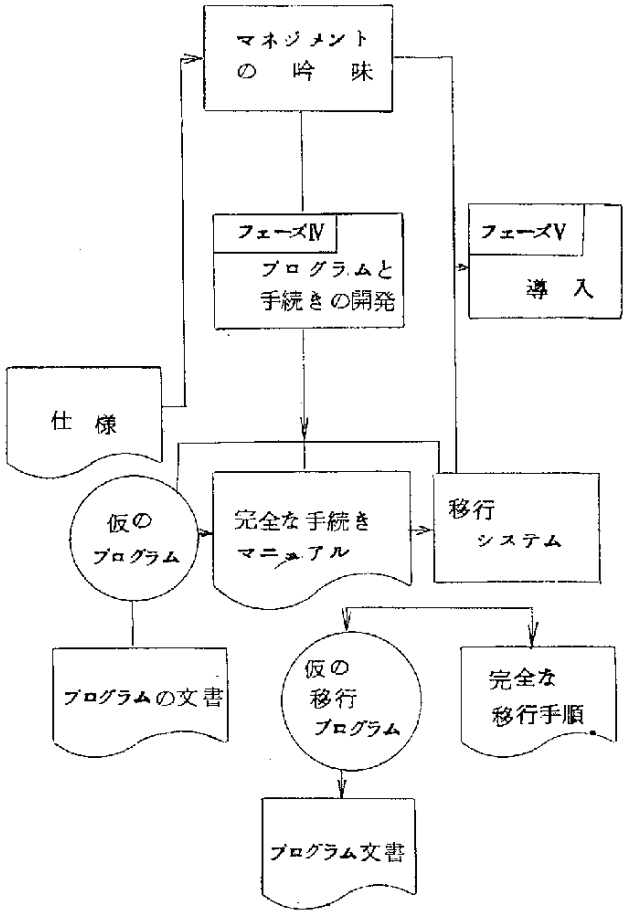
ストラクチュアルスタンダード	手　続　き　図　表	目　的
A　タスクリスト B　プログラム作成規則 C　テスト作成規則 D　手続き作成規則		1. すべてのプログラムをテストし文書にすること。 2. プログラムをシステムへ統合化すること。 3. 手続きマニュアルを完成すること。 4. 移行手順と計画を完成すること。 5. プログラム、手続き、移行努力を統合化してトータルシステムのテストを行うこと。 6. コストとスケジュールのすべての見積りの妥当性を調べ、必要があれば更新すること。
		責　任
		システム開発 すべてのプログラムと手続きを開発しテストする。 稼動コストを再評価する。 ユーザー 仕様とコストに関する変更を目的と見合わせて評価する。 移行時の訓練と教育の計画を立てる。 導入のスケジュールをたてる。 データセンター 移行時の訓練計画を立てるとともに、新設備の導入を助ける。 マネジメント 計画に対する進歩を評価する。

図８－１　フェーズⅣ：プログラムと手続きの開発

下位のストラクチュアルスタンダード

このフェーズで適用可能なストラクチュアルスタンダードは次の通りである。

- タスク・リスト
- プログラム作成規則
- テスト作成規則
- 手続き作成規則

1. タスク・リスト

購入者に対し、家の詳細設計を行う契約をする建設業者を考えてみよう。仕様書、価格、譲渡日は全て承諾を得ている。建設業者は、建築、電気、水道工事といった、設計のあらゆる面の詳細青図を作成しおえた。建設業者が利益を得る手腕はスケジュール通りビルを完成するために、業者のリソースをいかにうまく組織し、調整するかによる。彼は何時までに、どんな順序で何を各個人が達成しなければならないかを知らねばならないし、計画上の変動に対応できなければならない。例えば、4週間雨続きのこともあるし、重要な材料の引き渡しが遅れることもあるし、顧客が仕様書に変更を要求するかも知れない。（それは、もし彼が利巧な男であれば、顧客には費用が余計にかかって、彼には利益をもたらすだろう。）

今度は、情報システムの作成者である時点で考えてみよう。彼は一連のちゃんとした仕様書（プログラム、手続き、移行）をもっているし、彼自身も価格とスケジュールを決定している。彼はいまやスケジュールに合わせてプロジェクトを完成するように彼のリソース（プログラマー、手続き記述者、テスト時間要件といった）を組織しなければならない。彼は定め

られた時間内で各個人が達成できるものは何であり、サブルーティン、モジュール、プログラム、およびサイクルが作成され、テストされる順序を理解しなければならない。彼はまた彼の計画に影響をおよぼす多くの変動要因を処理しなければならない。第1に、テスト環境のコントロールは天気予報とほとんど同程度の確実さしかないし、第2に、要員の高度の移動（退職したり、配置転換で）で苦しむだろうし、第3に、仕様書の変更要求に苦しむだろう。ある場合にそれは、システムの構成要素が互いにうまく調和する（作成がより反復的で、確定した技術であり、その仕様書要件がもっとはっきり理解できるようになっている）ようになっているからであり、ある場合は、何かを行うよりよい方法が発見されるからであり、ある場合には、ユーザーがそのシステムを使う日が近ずき、そのシステムについてより深く検討しはじめるにつれ、ユーザーがシステム修正の要求書を作成するからである。

不幸なことながら、ユーザーは、一般的にいって、家の建設者がそうであるように、彼の突然起った妙案には法外な費用を支払らわなければといった条件がととのっていない。建設者と同じように、このフェーズで必要とされることは、タスクリストの作成である。このフェーズは開発サイクルの他のフェーズよりもより生産指向的である。一般にこれは次のようになされるべきである。

1. やるべき全てのタスク・リストを作成する。
2. タスク・リストに要員の要件を並べる。
3. タスク・リストを実行しなければならない順序に並べ変える。
4. 項目3の制約内で、平行的方法で実行出来る多くのタスクを調整してみる。

5. 予想最終日を計算する。それを強制された最終日と比較し、同様に、各リソースを予定されている平行業務と比較する。

6. 思い通りの最終日になるまで、各リソースと平行業務の均衡をとっていく。

これは多くの方法で行うことが出来る。大規模プロジェクトでは、ネット・ワーク図表とコントロール・システムが役立つ。ＣＰＭまたはＰＥＲＴ、またはこれらを基にした多くのパッケージについての議論は、利用出来る多くの参考文献をみるのが一番よい。プランナーにとって重要なことは、どのくらい詳細なものが必要かということである。これは作成上の問題であるけれども、実際に役立つプログラムと手続きを開発する上で、フェーズからはずれることは大変な罰である。このフェーズで作業する人々のまさに本質部隊でのある業務の遅れは、プロジェクト全体にわたる重大な遅れとコスト高をもたらすことになる。こうしてみると次の質問が起きてくる。このフェーズはどの程度の詳細さで計画すべきであるか？

３つの典型的なシステム開発要員の作業を検討してみよう。妥当な見積りは要員の作業の４０％がこの開発段階で費やされるとみなすことである。うまく計画すれば、平均で２週間の作業タスクが１つのある限られたアウトプットを生むように、作業タスクを定義することになるだろう。これらの要因が与えられれば、このタイプのシステム作業がもっている計画作業の数を計算することが出来る。この様子は表８－１に要約されている。

情報システムを作成するマネジメント技術は、単純および中程度の複雑として表中に分類されているシステムではなんとなく理解できるが、より大規模なシステム（そのようなシステムは次第に何度も起きつつある）を処理するのに必要なマネジメント技術は、これまでほとんど証明する機会

がなかった。

表８－１　プログラムおよび手続き開発のタスク計画（フェーズⅣ）

複雑度特性	例	全開発延年	延年（フェーズⅣ）	計画タスク数（フェーズⅣ）
単純	設定された販売および経費のデータ・ベースから得意先および商品利益の計算	2	0.8	20
中程度の複雑	一般的給与および労働分析	10	4	104
複雑	オーダーエントリーシステム（受注、伝票発行、売り掛け、在庫管理、販売分析）	100	40	1040

スケジュール化されねばならないタスクの数や、このフェーズで作成され、統合されねばならないプログラムや、手続きを検討することが唯一の結論となる。すなわち、フェーズⅣの開発作業では、一連の組み立ての流れを明らかにし、管理することを強調しなければならないということである。各組み立てグループは、そのグループより上位の、テストされた構成要素に統合しなければならない。より下位の組み立て（subassembly）は、開発フェーズの全域にわたって、出来るだけ早く行なわねばならない。

そうしないと下位組み立ての間の調整は非常に高価なものとなる。各開発プロジェクトはユニークではあるけれども、下位組み立てをつなぎ合わせていく（ｌｉｋａｇｅ）という確実な戦略は、一般的にいって、どれにもあてはまるし役立つものである。

A　プログラムの下位組み立て

下に示す順序は一般に、プログラム、またはプログラムの構成単位を、下位組立品につなぎ合わせるのに適用出来る。

1. 構成単位をつなぎ合わせる仕様書を検討し、何か変更があれば同時に個々の構成単位を変更する。
2. リンケージを設計する。
3. リンケージをコード化する。
4. リンケージを検証するテスト・データを作成する。
5. リンケージをコンパイルする。
6. 構成単位、リンケージ仕様書をテストする。

下位組立を行う仕事の順序は、もしそれが、サブルーティンをモジュールにまたは、プログラムをサイクルにつなごうとする場合には、同じである。これは、モジュールがシステムの本質的特性であるという、以前に行ったポイントを例証するものである。もし、モジュールがシステム設計の最初に認識されている場合は、このフェーズで完全な下位組立のリンケージがその結果として作られるだろう。もしモジュールが正確にちゃんとつながれていない場合には、下位組立を論理的に順序づけるのはむずかしくなるだろう。

B　手続きの開発

下記は手続き開発の秩序だった順序を説明するものである。

1. 手続き仕様書を検討し、いかなる変更も同時に行う。
2. フォーム設計の完了
3. 手続きの記載
4. ユーザーを抽出して手続きのテスト
5. 手続きに訂正を加える。

C 手続きとプログラムの統合

下位組立ての過程でのできるだけ早い時期に、手続きと移行の過程を明細に規定することによって作成されたテスト・データを使用すべきである。この意味は分析によってよりも、テストを通じて、プログラムと手続きの仕様書の一貫性を検証するということである。手続きは次の方法で下位組み立て的につなぐことが出来る。第1に、一連の手続きによってコンピュータ・インプットを作成する。これらは一見妥当なものであると思われるものが作成されたかどうか調べるテストをすべきである。第2に、手続きを経て作られたインプットは、できるだけ早い時期に、下位組立てのプログラムに対してテストすべきである。

2. プログラム作成規則

テストの容易さ、保守の容易さという、結果的にもたらされる利益をもつ、プログラムのモジュール化を達成することが望ましいということについてはほとんど議論されていない。しかしながら、プログラムがモジュール的に設計される設計標準を述べることと、それがプログラムグループ内で実行されるよう監督することは別の問題である。しかしながら、プログラムのモジュール化を促進するいくつかのストラクチュアルスタンダードがある。これらについてこれから述べる。

A　言葉の選択

ＣＯＢＯＬのような手続き向きの言葉は、プログラムの定義の際にすぐにいくつかのモジュールを作り出すということを知るべきである。ＣＯＢＯＬを例にとると、まさにプログラムの環境、データの定義、および手続きの部分がある。アセンブラー・レベルのコーティングでは、このモジュールの最低レベルでさえあいまいになる傾向がある。『もっと基本的なレベルに関していうと、一般にデータ・ベースとか、ファイル・マネジメントとして関係する言語では、システムは、コード化を構造モジュールにしようとする。ＭＡＲＫ Ⅳ を例にとると、プログラマーはファイル保守の過程を、各処理を定義するたびに、ファイルの保守行為の形でモジュール化しなければならない。彼はプログラムを、少なくとも一般的なモジュールにさらに構成しなければならない。例えば、ファイル保守行為、論理的および計算的処理、アウトプットの作成等である。ファイル保守でのように、モジュール化の構成はプログラマーに押しつけられる。論理的および計算的処理のためには、定められた各々の決定内で関連する全ての行為をグループ化しなければならない。アウトプットのためには、各アウトプットの全ての行為が１つの統一体として定義される。ファイル・マネジメント言語を、プログラマーが問題を速やかに定義し、コード化するのを助ける上で非常に強力なものにしているのは、このような特質である。プログラマーは、言語の諸特質によって、問題に対する賢明な構造的な解に導びかれる。

B　言葉のコード化規則

1. いくつかの構造的手法が有役である。『プログラムを少なくとも標準的特性の任意のモジュール数にモジュール化すること。例えば、エ

リアの確保、ファイル処理、レコード処理、アウトプットの作成、プログラムの終止等である。

2. プログラムを、どのモジュールに対しても、ある任意の大きさの制約のもとで、論理的に機能的な根拠に基ずいたモジュール化すること。他のモジュールについては無関係にコンパイルし、テストできるモジュールに、関連するサブルーティンを集めることが必要となるかもしれない。

C　詳細設計をより高位の設計に関連ずけること

これは詳細プログラム設計を、前の設計フェーズ中に行ったより高位のプログラム設計に関連ずけることを意味する。プログラム仕様書は各機能のインプットとアウトプットおよび関連する処理規則を記述することによって、実施されるべき一連の機能の相互関係を説明するものである。フェーズⅢで、これらはモジュールの相互関連および文章での仕様書に対する、処理行為への相互関係を明らかにする、概要プログラム流れ図によって記録されているモジュール化設計で関係ずけされる。このフェーズ（フェーズ４）でファイル図表が定義され、より低レベルの詳細設計が作成さるべきである。それから流れ図は原始コードと相互参照される。このようにしてモジュールを、次第次第に高位のプログラム仕様書から引きだすようにする。この処理の大半は、プログラマーの時間を節約するために機械化することができる。高位のモジュール流れ図は、ＡＵＴＦＬＯＷのような流れ図を作ることのできる言語の中のＮＯＴＥによって定義することができる。このようにして、プログラム、レコードのプログラム部分の構造を作る。ＡＵＴＦＬＯＷ、または類似の言語により処理された、詳細原始コードは、原始コードの流れ図表を図式的

に作成する。

3. テスト作成規則

私はもしシステム開発マネジャーが、フットボールのコーチのように、ゲームをふりかえって調べることが出来るならば、ある点で、苦しさのあまり立ち上り叫ぶだろう。" ここがわれわれが試合を失ったところだ "と。

良いテスト手続きには、いくつかの望ましい特性がある。それは以下のような点である。

- 各テストが充分考えぬかれていること。
- テストの進捗が測定出来ること。
- 1回のテストを短かくすること。
- テストとテストの間の準備を最少にすること。
- テスト・データがシステムのその後の保守用に使いうること。
- テスト・データ作成のための有効な方法があること。

これらの特性のいくつかを、テスト・ユーティリティ、またはモニターシステムのコントロールの下で全てのテストを処理させるようにすることによって、テストの環境に組み入れることが出来る。このようなシステムでは、テストのインプットを、テストのために必要とされる論理ファイルに続けて送り込み、これらの論理ファイルから、その後のプログラマーの分析のための情報をコントロール・モニターによって自動的に収集出来るように、全てのテストデータを保守可能な形式に組織づけることになる。このようなシステムは、附加的なものとして、テストデータをパラメーターにもとづいて作ったり、あるいは現存のデータ・ファイルから続々に送り込んだり、ある条件のもとでの保守をしたりするのに便利なフォーマッ

トにして抽出することによってテスト・データを作ったりする性格をもっている。また、承認を受けた多くのテスト・データは、テスト・モニターでコントロールすることが出来る。このタイプのユーティリティとモニターは、ある所与のテスト環境に積極的に作用をおよぼす上で、非常に有効である。

これは全て、テストをコミュニケーションを重んじたリモート・ジョッブ・エントリー環境を通じて実行させるようにすることによって、なし遂げることが出来る。よくあるのは、良いテスト・システムの特質というものは広く認識されてはいるものの、多くの場合に、その特質が手続き的に実行させるようにしなければならない場合には、その特質を無視してしまうということである。リモート・ジョッブ・エントリー・システムが意味する、プログラマーとコンピュータとを引き離すということは、計画および必要細目を達成させるように強制することであり、さもなければ有益なテストは全くなされないだろう。もしリモート・ジョッブ・エントリー・システムがうまく作動しないならば、その結果は、病気よりもはるかに悪い治療であることとなるだろう。

しかしながら、テスト自体は設計されねばならない。いくつかの気のきいた手法が、ストラクチュアルスタンダードが取りうる方向を示している。

1. テスト計画の基礎としての流れ図表を使い、次のことを行う。
   a すかしの上張り紙を使って、一組のテスト・データが設計されたら流れ図の箱のところに色線を引く。
   b 各々のテスト・データの組を追加するごとに、全ての流れ図の箱が少なくとも1回は通過するまで、別の色線を反復する。
   c テスト・データの各組は、インプット・レコードの内容によって注

意深く記述すべきであり、テストごとの、個々のデータに対する、アウトプットの結果を予想すべきである。

d　次にテスト計画は、“赤テスト”、それに続く“青テスト”、“黄テスト”等と処理することからなる。

e　テスト状況の評価は％で表わされ、その場合、９０％とは流れ図箱の９０％がテスト・データで主首よく通過したことを意味する。

2. 類似の手法は、プログラムの全ての決定点を印刷し、次に決定点の各々のテスト・データの記録をリストするというやりかたである。もう一度最初の手法に戻っていえば、各々のインプット・テスト・レコードとそれによって作り出されるべき結果は、テストに先だって、はっきりと説明さるべきである。

この両手法およびこれらに類似した手法を使えば、非常に少数のテスト・データで充分に説明できるようにするし、無理なく、プログラマーが、テスト実行前に予想結果を確認するために、自分のプログラムを注意深くデスク・チェックするようにさせるだろう。

これまでの検討内容は、サブルーティンからモジュールへ、さらにプログラムへという、１単位または下位組立テストに一般的に適用することができる。これ以上のレベルのテストでは、次のことに関心を払うべきである。単体は適切に結合しているか？（すなわち、テストによって論証可能な下位組立間のデータの流れのことである）およびシステムの手続き書の要求事項に基ずいて、外部的に作成されたデータは、仕様書の問題を示しているか？　これまで述べてきた手法はいくつか適応性をもっているが、システムを有効に働かすことが出来るテスト・データを作成する。最少の手続を作り出すことのためには、さらに他の手法が必要である。

4. 手続き作成規則

手続きを文書化する人に、手続きをシステムのユーザが容易に理解できるようなレベルで彼の文書を書かせるための２・３の手法がある。単にパラグラフに分けて書かれたマニュアルは、それに従って何かに使おうとする場合には制約が多すぎる。パラグラフ形式の文書をもとにして、使用する場合のことを考慮した、マニュアル構成の別の方法がある。これらの手法はマニュアル作成にあたっては適切であり、以下に述べる。

A　演劇脚本形式

この手法は、複数の人または部門で、各々異なる責任をもっている場合に、それに従って行う手続きを説明する場合に有益である。パラグラフ形式を使う代りに、文書があたかも演劇脚本のように作成され、役者名がつけられるところに従業員名や部門名がつけられ、対話のかかれるところに指示がか条書にされる。図８－２はこの手法を使って書かれた短かい指示である。それらは複数の人がいかにして１つの形式にまとめあげられるかを示している。

主題：　給与カード、９４９を完成する方法

責　任	行　為
新従業員	1. 第1行目に名前と住所を書く
	2. 第2行目に社会保障番号を書く
	3. カードの一番下に名前を書く
	4. 監督者にカードを渡す
監督者	5. 第3行目に従業員の業務を書く
	6. 給与部にカードを送る
給与係	7. 第4行目の従業員の認識番号を書く
	8. 給与ファイルにカードをファイルする

図８－２　演劇脚本形式

B　分割頁方式

この手法は、１つの特定の業務が行なわれる際に、１グループまたは１つのタイプの従業員がそれに従う詳細指示、例えば給与管理の合計時間値を書いたり、新従業員人事記録カードに書き入れるといった指示を記載する場合に使うのに有効である。それはまたある特定の事項がどう処理されたかを説明するのにも有益である。

指示は概要で始まり、それに続いて２頁の左側に一般指示、右側に特定指示がくる。これはその仕事をよく知っている担当係が、為さねばならないことのチェック用として左側を使い、彼が確信がない項目をすみやかに見いだすのに右側を使えるようにする。頁の右側の特定の指示は、新しく来た係を訓練し、手続きが１歩１歩注意深く守られるのを保障す

るために使われる。

図８－３はこのタイプの文書の例であり、従業員の出生年月日がいかに記入され、人事システムに転記されるかを説明するものである。

	出生年月日
概　要	従業員の出生年月日は、新規従業員様式にもとずいて人事係によって、人事システムに入れられる。もしそれが間違っている場合には、人物概要変更様式で変更される。 項目に対するいかなる変更であっても、新人物概要を印刷するようにする。
1. 新規従業員様式に出生年月日を記載する	A．出生年月日を、第６行目の第４欄から９欄に記載する。 B．月月　日日　年年　と記載する。
2. 人物概要変更様式、カードタイプ３１Cで出生年月日を変更する。	A．出生年月日を２９欄から３４欄に記入する。 B．月月　日日　年年　と記載する。

図８－３　分　割　頁　方　式

C　相互参照説明図

この種の図表は手続き記載者にとっても、保守システムのアナリストにとっても有益な道具である。手続き記載者を必要とするのは、彼のマ

ニュアル作業流れ図とプログラマーのシステム流れ図とを統合する必要性からである。各マニュアル作業流れ箱には、丁度システム流れ図の各プログラム箱に、プログラム処理番号が示されるのと同じように、手続き番号が示される。これは次のような効果をもつ。

1. 手続きが各項目ごとに書かれるのを保証する。
2. プログラムに変更がなされる場合、保守プログラマーは変更によってどの手続きが影響を受けるかが速やかに知り、手続き記載者に知らせることが出来る。

第9節　導　　入

概　　要

このフェーズでは大量の稼動データの下でシステムを検証し、次に旧システムから新システムへの移行を完了する。それは図9－1で図表化されており、この章でさらに詳しく述べられている。

インプット

このフェーズのインプットは、新システム、および新システムを実行するのに必要なプログラムと手続きであり、新システムへの移行に必要なプログラムと手続きである。更に、インプットにはマネジメントの吟味によって公式に要求される変更項目もある。

アウトプット

このフェーズからのアウトプットは新システムを実行するのに必要な、証明ずみのプログラムと手続きである。重要なことは、移行プログラムと手続きはもはやいらないということである。

下位のストラクチュアルスタンダード

定義されたストラクチュアルスタンダードはタスク・リストであり、分割評価である。

1.　タスク・リスト

このフェーズは2つの別々のサブフェーズをもつ。第1に、システムの稼動能力が証明されねばならない。前のフェーズでは、システム統合テストで述べたように、理想的な、研究室的条件の下で、システムが稼動可解

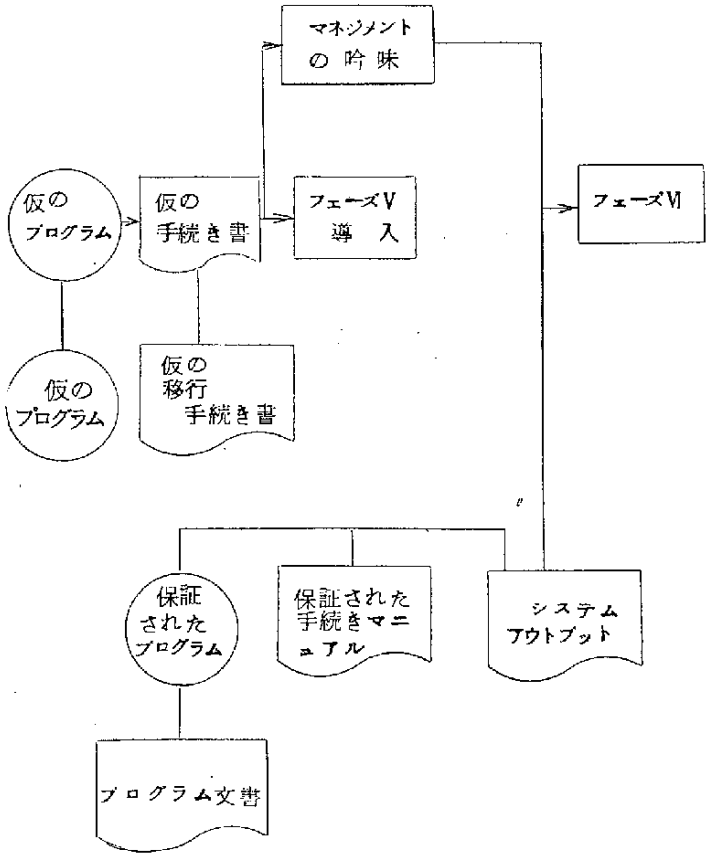
ストラクチュアルスタンダード	手　続　き　図　表	目　　的
A　タスクリスト B　分割の評価		1. 約束された機能を実行するシステム能力を調べること。 2. すべての操作を新システムに移行すること。 3. 必要に応じてプログラム、手続きおよび文書を更新すること。 **責　　任** ユーザー 少量のデータでシステムをテストする。 仕様上の変更を要求する。 修正がなされたあとで、システムへのデータをます。 コスト評価を続ける。 データセンター 少量のデータでシステムをテストする。 仕様上の変更を要求する。 修正がなされたあとで、システムへのデータをます。 システム開発 ユーザーやデータの要求に応じて仕様を修正し、プログラムや手続きを変更する。 変更にかかるコストを明らかにする。 マネジメント 計画に対する進歩を評価する。 新システムの結果を評価し始める。

図９－１　フェーズＶ：導入

であるということを証明したそれは次のような特性をもっていた。

1. システムによって定義された標準手続きに従って、システムへのインプットの作成。
2. システムによって実行される標準手続きに従って、エラー分析の実施。
3. 稼動動向からの、システム稼動をテストするための"生"データの利用
4. システムの実行に基ずいた、プログラムと手続き改善。
5. 複数の部門からの多量のデータを分割する際の問題点を指摘するための分散分析に使える、正確な稼動効果推定値の確立。

このフェーズで、システムが"実世界"条件の下で作動可解であるということを証明しなければならない。これは平行して稼動テストを実施することによって行うことができる。平行して稼動させる場合に、新システムと旧システムの両方が、旧システムをとりさっても安全であるということを示す。充分な証拠があるまで、特定のその場所、または複数の場所で稼動される。システム・テストの結果、理想的な状況の下で実行した際に、そのシステムが、生データで適切に作動あることを示す場合であっても、それは実世界のほんの近似にしかすぎない。システム・テストは、スケジュールに合わせねばならないという制約とか、オペレーターの特質と対立すべきものではない。平行した稼動は、稼動現場でのシステムの実行能力を評価し、一方では、稼動現場をシステムの失敗の危機から守るものである。

現実の状況の下で、システムがいかに作動するかを調査する第3の方法は、パイロット稼動テストである。これには、システムの最初のユーザーとして、1ケ所または複数ケ所の現場をとりあげる。ユーザーと協同で得

た経験から、より極端な分割を計画する。

システム次第で、これらの手法を組み合わせて使用する。これらのシステムと本来のシステムの間の相異が多きすぎるため、平行した稼動ができないシステムもありうる。システム・テストから、平行した稼動またはパイロット稼動なしで高い割合の分割へ進むことは、"捨身でかかる"アプローチとして説明されている。この手法の危険が、いかなる大規模なシステムにとっても、いかに危険なものか保証するのは困難である。

2. 分割（Cutover）の評価

実行すべきシステムの能力をテストするために、何段階かに分けてテストを行う手法の利益の1つは、分割の際に、システムがいかにうまく作動しているかを正確に確認するための、有益な情報が作成されるということである。このように、システム・テスト、平行テスト、パイロット・テストの全てが、主首よい分割をするのに必要な情報にもとずいた標準を作成すべきである。分割の例は、ファイル移行、インプットの準備とそのエラー率、システム、インプット・エラー率、コンピュータ利用率等である。これらの領域の各々での計画利用数字が、分割の各ステップについて作成され得るし、そのステップが達成されるにつれて、実現値と比較することができる。このようにして、もしシステムの運用が、ジィオパーディ（jeopardy）であるならば、分割過程の初期の段階で発見することができる。

第１０節　システム移動

概　要

このフェーズでは、稼動面での要求とデータ・センターの環境からの要求が指示するように、システムを修正する。このフェーズでは、また、費用を正当化しうる場合には、いつでもシステムの稼動効率を改善しようとする。このフェーズが図１０－１で図表化され、さらにこの節の内容に合わせて詳しく説明する。

インプットとアウトプット

システム開発の他のフェーズと違って、システム稼動は、始めと終りをはっきり定義していない。そのインプットはシステムであり、そのドキュメンテーションも、そのアウトプットと同じくシステムである。またそのインプットは、ビジネス条件、技術、マネジメントの政策、政府の規制、またはその他の理由に伴って、その形が変るかも知れない。

下位のストラクチュアルスタンダード

このフェーズでの適切なストラクチュアルスタンダードとは、変更とシステム改善の管理である。

1. 変更の管理

稼動システムには、保守費用のライフ・サイクルがある。先ず第１に保守費用は２つの要素からなる。

1. システムから不安定要因（またはムシ）を取り除く。
2. システムを変更条件または追加要件に合せて変更する。

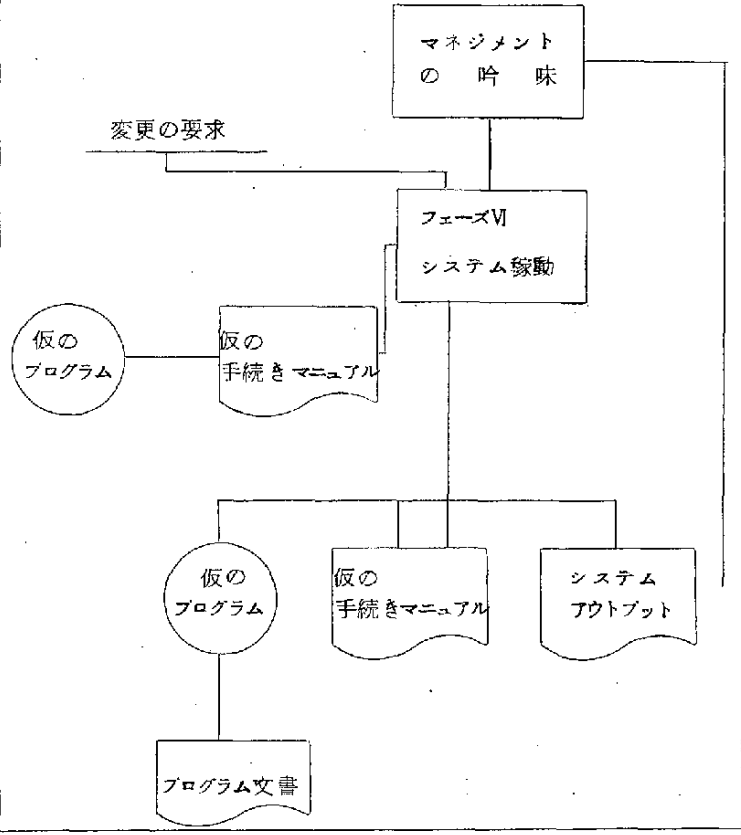
ストラクチュアルスタンダード	手　続　き　図　表	目　的
A　変更の管理 B　システム改善		1.　アプリケーションやシステム技術の状況からの変更要求に従ってシステムを修正すること。 2.　システムをより信頼性が高くて効果的にする努力を続ける。 責　任 ユーザー 新しい稼動条件、新しい報告書あるいは改訂された報告書、新しい価格政策等に合うような変更を要求する。 データセンター 稼動効率を上げることに努力を集中する。 システム開発 要求に応じて仕様書を修正し、プログラムや手続きを書きかえる。すなわちプログラムメインテナンス マネジメント システムの収益性を評価する。 データ処理の総コストへの効果を評価する。

図１０－１　フェーズⅥ：システム稼動

一定期間後、不安定要因が一般的にいって、システムから取り除かれる。（決して完全ではないが）そして要件の変更が問題の基点となる。ある時点において、変更要件は、システムおよびその設計概念が、本来考えていたものとは異なる物理的環境を強引に取り入れようとさせるために、ますます大きなものとなる。新しい変更がシステムに導入されるにつれて、その変更は、訂正しなければならない不安定要因をまた作り出す。これらの概念を図１０－２に示す。

図表をみてみると、ある時点では、そのシステムを物理的環境をよりよく説明する他のシステムと取り換えた方が効率的になる、いということが解る。

課さねばならないストラクチュアルスタンダードは、システムになされる変更のタイプと、変更の頻度と、変更の費用とを記録したものである。そしてこの変更費用は、システムが、そのライフ・サイクル最終の段階になったとき、その費用が明らかになるように、稼動費用と相互関連づけられねばならない。

2. システム改善

システムの技術が未熟であるという証拠の１つは、類似したプロセスの概念が利用されないということである。新製品が工場で生産に入る場合には、その製品は開発グループの手を離れ、工場技術者が、標準製造費のもとでその製品を引きつぐとともに、製品のコストと信頼性に対する改善は続けていくことになる。このことは、もしデータセンターが工場であり、システムが製品であるとするならば、システム稼動費を削減し、その信頼性を増大することが、真に組織の奨励要因になるから、データ・センター

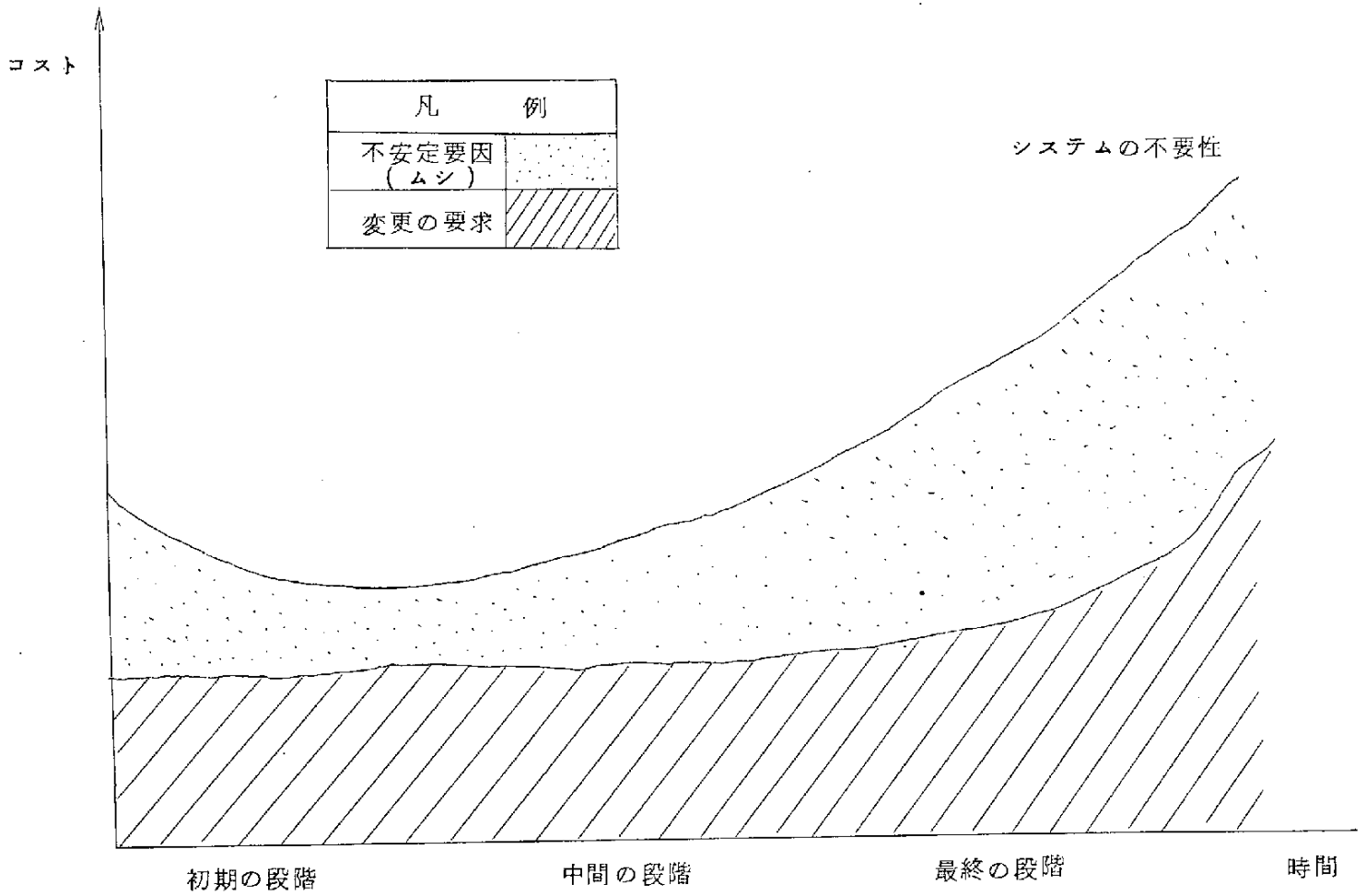


図１０－２　稼動システムのライフサイクル

がシステムを保守すべきである。

明らかにストラクチュアルスタンダードはシステム組織（理想的にはデータセンター）における1グループに、システムが導入された後で、システムを改善するために特定の責任を与えることであり、そのグループに良い仕事をするための刺激を与えることである。これを行なう1つの方法は、改善を行うグループが、彼等の改善努力によってなされた節減額の何割かをデータセンターに請求させることである。このようにして、もし彼等が1年のコンピュータ時間で、20000ドル節減するというシステム変更をした場合には、その何割かが彼等のものとなり、残りの何割かがデータ・センターのものとなるようにすべきである。このような計画によって、彼等はシステムを絶えずより効率的にする刺激を与えられるだろう。

第１１節　ストラクチュアルスタンダードの設計

概念の要約

われわれはこの本の中で次のような考えを開発してきた。

1. システム開発が、ピーター・ドラッガーの言葉を借りれば、“知的作業”であるので、もともと困難な業務であり、われわれは、高質の知的作業を妥当な費用で生産する経験が浅い。
2. われわれが情報システム開発と対比するのに最もよいものは、製品開発のサイクルである。
3. システム開発上、われわれが認識すべきいくつかの基本的原理がある。これらの原理（原則としてここでは述べられている）は、開発サイクルに統合さるべきである。
4. ストラクチュアルスタンダードは人々に、ある定性的、定量的基準がアウトプットの質を保証しうるように、アウトプットが充分に標準化されている品質の高い作業を考え出すようにするための、標準モデルである。
5. システム開発のプロセスは、主要なストラクチュアルスタンダードまたは開発システムのモデルである。それらはさらに細かいストラクチュアルスタンダードに分割できる。これらのストラクチュアルスタンダード開発過程を導びき、管理する手段を提供する。ストラクチュアルスタンダードを設計するガイドラインがあるかどうかという疑問が生ずる。
6. ここで開発されたストラクチュアルスタンダードは、ストラクチュアルスタンダードの設計への有益なガイドラインになりうる、一般的範疇に入るものである。これらのガイドラインを以下に説明する。

構造化特性としての推定

厳格に推定することが開発過程に課せられる場合はいつでも、その開発過程が妄想からときはなたれ、より確固たる足がかりを作っていく傾向がある。コスト評価様式（図５－３、８２～８３）は、ストラクチュアルスタンダードとして評価案を使用した１例である。同様に、プロトタイプ・システムのシミュレーションは、実際に役立つシステムに先だって、有効性を評価するために案出したものである。

この本では検討されている２・３の例以外ではふれていないけれども、評価過程が提案システムの全ての各々の部分のコストをカバーするために、フェーズⅡ、フェーズⅢにも同様に拡張されるべきである。例えばコア記憶装置の必要量、プログラムの大きさ、処理件数、ファイル作成と移行などである。詳細見積りの開発では、"実現可能な限り多くの代替案を考慮せよ"という原則をしっかり守るようにする。なぜなら詳細見積りは、提案した代替案を最終的に評価するための、冷静な論理的基礎を提供するからである。

構造化特性としての概要（ｏｕｔｌｉｎｅ）様式の使用

この本ではストラクチュアルスタンダードとして、概要または様式の使用についていくつかの例が書いてある。実行可能性研究報告書システム仕様報告書は、概要の使用がいかにしてプロジェクトを構造化しうるかを示している。同様に、ＡＤＳシステムで示す様に、相互参照マトリックスへの設計データを減らす様式の使用は、貴重な構造化手段である。最近出版された２冊の本は、そのほとんどのページを、システム開発の構造化手段として様式と概要を使用するテーマに費やしている。

構造化特性としての図表の使用

われわれはいくつかの図表形態をとりあげてきた。開発過程を計画するネットワーク図表、システム全体のデータの流れとコントロールの流れの相互関係を示す流れ図、そしてプログラム流れ図等である。これらの事の全てにおいて、図表は、計画が行動の重要な部分になる場合に、何かを行うための計画にほかならない。ピーター・ドラッガーは、知的作業においては、計画と行動は分割することができない、と「断絶の時代」の中で指摘している。上記の図表の形態は、この特性の優れた事例であり、それらの事例がなぜ有益なストラクチュアルスタンダードであるかを説明している。

要　約

私は、この本の結果として、システム開発者同志で、開発プロセスを構造化するアイデアの交換があり、われわれの仕事のより困難な興味もある分野で、より広範囲に使いうる手法を構造化したものが、実際に形式化されたことを期待するものである。

附属資料 2

プロジェクト・ライフ・サイクル

これは Sun Oil Company (Philadelphia U.S.A) が自社用に開発したマニュアル "Project Life Cycle" を全訳したものである。

プロジェクト　ライフ　サイクル

１９７０年１０月１日

題　目：プロジェクト　ライフ　サイクル　タスクフォース　レポート

年月日：１９７０年１０月１日

提出者：J．P．マックール

タスクフォース　メンバー

H．B．ディーツ	T．タンソール
R．L．フィンク	J．F．ソーントン
F．レッシュ	E．B．ウォーウィック
J．L．ロングッティー	R．C．ツェールク
W．A．シュパイツ	

本プロジェクトの目的は、我が社のＳ＆Ｃ部門（System & Computers Division）ならびに各ＯＳＤ部（Organization & System Development Departments）が種々のプロジェクトに取り組む際に標準となるひとつのフレームワークを提供することでありました。このようなフレームワークは、プロジェクトに関与する全ての部門間の良好な意思疎通と協力関係のために、なくてはならないものであります。

本プロジェクトの目標は、依拠すべき標準としての「プロジェクト　ライフ　サイクル」を関連部門間の意思疎通の方式と共に、新たに設定することでありました。

プロジェクトの範囲としては、「プロジェクト　ライフ　サイクル」全般についての定義を与えること、サイクル内の各フェーズを定義すること、お

よびフェーズ内でのいくつかのタスクを指示することがその内容として決定されました。各フェーズを「如何に」行なうかの問題は範囲外として、現在進行中の別途プロジェクトで取り扱われております。

プロジェクトを遂行するにあたり、J．P．マックールがプロジェクトリーダに任命され、またS＆D、本社OSD、製品部門OSD、資源開発・製造部門OSD、南西部電算センタ、北東部電算センタからの各代表者によるタスクフォーズが組織されました。各代表者は、「プロジェクト　ライフサイクル」に関する、所属部門からの色々なアイデアや考え方を提出しました。タスクフォースは「プロジェクト　ライフ　サイクル」の草案を数種作成、配布、検討、評価し、内部のミーティングで意見が一致した種々の修正をこれに加えることも行いました。

これらの最終的成果として生まれたものが、この「プロジェクト　ライフサイクル」ならびに「付録」であり、「付録」には、「プロジェクト　ライフ　サイクル」の諸部分を明確化するに必要なかぎりの詳細が示されています。このプロジェクトの最終所産は、タスクフォースの全員が検討し合意したものであります。

「プロジェクト　ライフ　サイクル」のオリエンテーション　プログラム完了後、新たに着手されるプロジェクトは、すべてこのフレームワークに沿って実施されるべきであります。すでに実施中のプロジェクトについては、オリエンテーション期間中に、段階的に新しい方式に合わせてゆくことになるでしょう。

この方式を生み出すにあたって発揮されたタスクフォース　メンバーの努力に対しては、心から感謝したいと思います。また初期の段階でのD．W．フィッシャー氏の貢献と、D．B．ウィリアムス嬢の優秀な秘書としての助

力は、本文書を作成するにあたって、大きな援助でありました。

J．P．マックール

プロジェクト　ライフ　サイクル

序　　言

「プロジェクト　ライフ　サイクル」の目的は、システム、コンピュータ部門（Division）および各部門内のＯＳＤ部（Department）におけるリソースの組織ならびに運用を改善することである。

「プロジェクト　ライフ　サイクル」の目標は、上述の各部門がプロジェクトにおいてユーザを援助する場合に準拠すべき一連のフェーズを明確に区別し規定するところにある。これらのフェーズは、プロジェクトがサイクルをなすように、そして必要があればさらにそれを繰返すように組み立てられている。いくつかのフェーズを設けることにより、プロジェクトの特定時点で、管理者側がそれまでの活動を点検し、承認を与えることができるようになる。各フェーズは定義され、内部的な相関関係もここに示される。「付録Ａ」は、各フェーズに関する詳細であって、それぞれのフェーズの意味するところが何であるかという問に対する回答になっている。

「プロジェクト　ライフ　サイクル」が取り扱う範囲には、Ｓ＆Ｄ、各ＯＳＤその他類似部門がシステム　アナリシス、デザイン、オーガニゼーション　アナリシスおよびコンサルティング業務に関与する際、活動の標準フレームワークとしてこれを適用する場合のその方法も含んでいる。「ライフ　サイクル」の意図は、サンオイル全社におけるこの種のプロジェクトの計画と相互間の調整を助けることであって、サイクル全体を通じてユーザ側の参加を予定している。「ライフ　サイクル」はまたプロジェクトの所要時間、経費の報告や複数プロジェクトの計画等の基礎として利用することもできる。

「プロジェクト　ライフ　サイクル」は、それ自身でプロジェクト　コント

ロールの機構となることを意図するものではない。どのフェーズについても、それを「如何に」行なうかという問題は、この「プロジェクト　ライフ　サイクル」のなかには含まれず、むしろ別の作業の題材として取り扱われることになろう。

以下は「プロジェクト　ライフ　サイクル」を適用するにあたって有益と考えられるガイドラインである。

A．プロジェクトのいくつかの側面のうちで、「プロジェクト　ライフ　サイクル」では取り扱われていないものがあるが、これらは、その特定プロジェクトに関与する管理者が判断し決定したほうがふさわしいと思われる性格のものである。各フェーズに誰を割り当てるか、プロジェクト担当者間の報告関係はどうするか等の人事的な判断がその例としてあげられよう。

B．「プロジェクト計画点検（Project Planning Revies Activity)」（4－a参照）は、「プロジェクト　ライフ　サイクル」のフローに影響を及ぼすものである。ただし、プロジェクトの規模が小さい場合は、これを省略してもよい。

C．あるひとつのフェーズ内で、数個のタスクを同時に行なうことは可能である。しかし、1プロジェクト内の数個のフェーズを同時に行なうことは不可能である。

D．「プロジェクト　ライフ　サイクル」で重要なひとつの考え方は柔軟性ということである。本「ライフ　サイクル」は、この「序言」でふれた部門が行なうすべてのプロジェクトに適用されるよう意図されている。しかし、柔軟性という考えの中には、個々のフェーズ内の細かなタスクに関しては、プロジェクトを管理する側の裁量に一切任されるということが含まれている。小規模プロジェクトの場合、いくつかのフェーズをきわめて短

期間に了えてしまう等のバリエーションは予め想定されている。しかし、大規模プロジェクトにあっては、あるフェーズが長い時間を占め、また本書に述べるよりもはるかに詳細な内部の計画を要することになるかもしれない。

概　　要

第1フェーズ　事前検討(Initial Investigation)

「事前検討フェーズ」の目的は、ユーザと協働し、次のフェーズである「予備調査」へ進むことが妥当かどうかを決定することである。事前検討の内容は、プロジェクトのタイプによって大きく変わってくるが、時間的には非常に短いフェーズであるべきである。このフェーズの最も重要な最終成果は、はたして、問題あるいは可能性が存在するかどうか、研究対象として採り上げるべきかどうかに関する判断である。答が「然り」である場合は、何故、また如何に次の「予備調査フェーズ」を実施するのかを明らかにした調査範囲説明書(survey scope paper)、およびそのための作業計画を作成することが望ましい。

第2フェーズ　予備調査(Preliminary Sarvey)

「予備調査フェーズ」の目的のひとつは、前のフェーズで存在するとされた問題あるいは可能性に対して輪郭を与え、これを検討することによって、「はたしてプロジェクトが必要か否か」のユーザによる判断を助けることである。もうひとつの目的は、いまだ検討段階にあるプロジェクトに対するアプローチを、ひとつまたはそれ以上提出することである。✳「予備調査」は、検討中のこのプロジェクト全体からすれば相対的に短期の努力であるべきである。このフェーズの主要な最終成果は、プロジェクトに対するいくつかの代替的アプローチ、各アプローチ毎の所要人員、コスト、利益の概算見積、推薦アプローチおよび基本範囲説明書(charter scope paper)であろう。基本範囲説明書では、何故、また如何にそのプロジェクトを実施すべき

かが概括的に提示される。これらに付け加え、第3フェーズ「計画」のための概略的作業計画が作成される。

第3フェーズ　計画（Plenning）

「計画」フェーズの目的は、第4フェーズの計画を詳細に立て、またプロジェクトの以降の部分について全般的計画を作成することである。これまでの説明でわかるように、初期のごく概略的な計画作業は、「予備調査フェーズ」である程度なされているが、この作業は、プロジェクトに対して承認が与えられ、正式にそれが存在するようになるまでは最小限にとどめられるべきである。承認が与えられると、この「計画フェーズ」において、「プロジェクト　ライフ　サイクル」の対象領域が拡大され、その特定プロジェクト内の各フェーズのタスクが規定されるようになる。これに加え、プロジェクト担当者の顔ぶれの決定、詳細作業計画の作成が行なわれ、プロジェクトの基本範囲説明書が更新される。フェーズの主要アウトプットに含まれるものとしては、プロジェクトの概略的作業計画と「研究・分析・予備設計フェーズ」の詳細作業計画がある。

※　ここでは、S＆D部門や各OSD部の行なう事前検討や予備調査には、正規のプロジェクトに発展しないまま終わるものもあることが想定されている。というのは、「予備調査」直後の点検の結果、いくつかの問題あるいは可能性は、サシ　オイル社にとって当面さして重要でないか、あるいはあまり良い成果が期待できないということになる可能性があるからである。第1および第2フェーズでの時間と、費用を最小限にとどめることの意義は、ここにあるわけである。この点検作業は、「予備調査フェーズ」と「計画フェーズ」との間に行なわれることになろう。すべての予備調査は、計画

調整のため本社ＯＳＤのシステム　プロジェクト計画課（System Project Planning Section）へ報告されなければならない。

第４フェーズ　研究・調査・予備設計（Study Analysis and Preliminary Design）

本フェーズの目的は、問題あるいは可能性を研究し、妥当であれば現行システムを研究し、解決策あるいは新システムの要件を決定、分析すること、そして、最終的解決策についての予備的設計作業を行なうことである。フェーズのアウトプットは、問題あるいは可能性の明瞭な確定、および推薦された予備設計についての意見の一致である。範囲説明書および作業計画の修正が必要に応じて行なわれ、「設計フェーズ」のための詳細作業計画が作成される。

第５フェーズ　設計（Design）

「設計フェーズ」の目的は、問題を解決するか、あるいはプロジェクトに託された可能性を実現するであろう方策を具体的に規定することである。新しい組織、作業方法あるいはシステムのための仕様が全面的に設計、文書化されて、予備設計は解決策の余すところのない設計へと発展させられる。フェーズのアウトプットは、問題解決の方法あるいはプロジェクトに託された可能性実現の方法を指示するため作成された設計に対する確定的合意である。範囲説明書および作業計画の修正が必要に応じて行なわれ、「開発フェーズ」のための詳細作業計画が作成される。

第6フェーズ　開発（Development）

「開発フェーズ」の目的は、机上の解決策を現実化するために必要ないくつかのステップを実際に踏むことである。そのなかには、コンピュータ・プログラミング、社内のある部分のための組織の新設、ＯＲモデルの考案、訓練プログラムの作成等が含まれよう。フェーズのアウトプットは、全面的に開発され、いつでもテストおよび実施へ進めるようになっている解決策である。範囲説明書および作業計画の修正が必要に応じて行なわれ、「テストフェーズ」のための詳細作業計画が作成される。

第7フェーズ　テスト（Test）

「テスト　フェーズ」の目的は、開発作業の結果を現実的状況あるいは人工的状況においてみて、はたしてこれまでの努力の結果が、本当に問題の解決となっているか、あるいは可能性の実現に向かっているかどうかを判断することである。このフェーズは、その特定プロジェクトの主題が何であるかによって大いに違ったものとなりうる。フェーズのアウトプットは、全面的にテストされ、その修正も含めて導入段階へ進むことができるようになった解決策もしくは新たな作業方法である。範囲説明書および作業計画の修正が必要に応じて行なわれ、「実施・評価フェーズ」のための詳細作業計画が作成される。

第8フェーズ　実施・評価（Implementation and Evaluation）

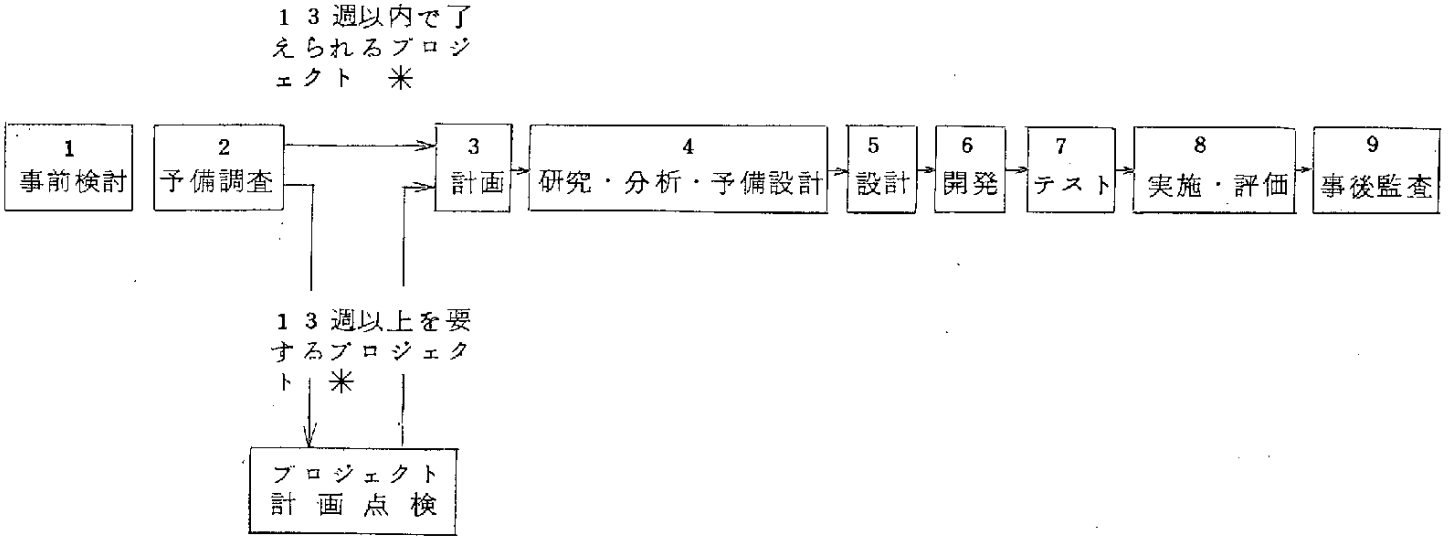
「実施・評価フェーズ」の目的は、サン オイル 社内で、プロジェクトの成果を以って問題を解決し、欠陥を是正し、可能性を活用することである。これに加え、実施初期の結果を評価することにより、解決策あるいは新作業

方法が設計に際して意図されたことを確実に実現するようにすることも目的となる。フェーズのアウトプットは、評価の結論にもとづく調整を加えた実際用の解決策あるいは作業方法である。「事後監査フェーズ」のための詳細作業計画が作成される。

第9フェーズ　事後監査

「事後監査フェーズ」の目的は、プロジェクトの運営上、財務上の一貫性ならびにプロジェクトの成果たる解決策の効果を評定することである。アウトプットとしては、監査報告書が作成され、これには必要に応じ、将来にそなえての勧告が添えられる。

プロジェクト　ライフ　サイクル　　フェーズ　ダイアグラム



＊　この数字は仮の標準である。

プロジェクト　ライフ　サイクル
付録　　A
各フェーズの主要ステップ、最終成果ならびにガイドライン

第1フェーズ　事前検討

A．主要ステップ

1. ユーザから要請が寄せられるか、S & D部門あるいはOSD部に対して最初の接触が行なわれる。
2. ユーザあるいは問題に関与しているアナリストによって、事前検討の要請が正式に文書化される。
3. 問題領域を定義するためにふさわしい方針を分析する。
4. ユーザと協力して「予備調査」を行なうための計画を立てる。
5. 調査範囲説明書を作成し、そこに「予備調査」の目的、目標、範囲、調査方法、生み出すべき最終成果、ガイドラインを規定する。

B．最終成果

1. 「予備調査」を開始すべきか否かの決定。
2. 調査範囲説明書。

C．ガイドライン

1. 「予備調査」開始の承認は、プロジェクトの対象範囲に相当する権限を有する者が与えるのでなければならない。これは、まずなによりもユーザ側の責任事項である。
2. 本フェーズの目的は、はたして問題あるいは可能性が存在するか否か、予備調査を行なうべきか否かについて意見の一致を得ることであ

って、問題を解決したり、問題あるいは可能性と取り組む方法を規定したりすることではない。

3. 事前検討は特別の事情のないかぎり、1～2週間で終えるべきである。

第2フェーズ　予備調査

A. 主要ステップ

1. 問題あるいは可能性がさらに突込んで検討され、一層詳細に文書化される。
2. プロジェクトに取組むための、1つまたはそれ以上のアプローチの概要が作られる。
3. 各アプローチ毎に、以下の点についての大まかな予測が行なわれる。

   所要人員の量およびタイプ

   プロジェクトの全体的費用

   予想される利益

   全体の時間的関係およびプロジェクトのスケジュール
4. 基本範囲説明書が作成され、プロジェクト全体における以下の諸側面について述べられる。

   目　的

   目　標

   範　囲

   アプローチ

   最終成果

   ガイドライン

B．最終成果

1. 基本範囲説明書。
2. より完全な問題あるいは可能性の確定。
3. プロジェクトに対する複数のアプローチの各概要。
4. 各アプローチ毎の所要人員、コスト、利益の概算見積。
5. 各アプローチの評価、あるアプローチの推薦。
6. 次のフェーズ、「計画」のための概略的計画。

C．ガイドライン

1. しかるべき能力を備えたユーザ側および非ユーザ側双方の職員からなるタスクフォースによってこの調査を行なってもよい。
2. プロジェクトはこれからのものであり、それについての見積は極く大ざっぱなものであること、以後における研究の結果によって変更されうるものであることが了解される。
3. 予備調査は、ＳＰＰ課の定義によって小規模プロジェトとされるものよりも長い時間を要すべきではない。標準以上に長期にわたる予備調査は停止され、更にそれを細分する必要があるかどうかの評価を受けるべきである。検討中の問題は、単一のプロジェクトで取り扱うには大きすぎるかもしれないということである。

第３フェーズ　計画

A．主要ステップ

1. 研究・分析・予備設計フェーズのための人選・配属が完了する。
2. プロジェクト担当者間の報告関係の形態が選ばれる。
3. プロジェクト担当者に対するオリエンテーションが行なわれる。

4. 次のフェーズのための詳細作業計画、およびプロジェクト全体の概略的計画が立てられる。

5. プロジェクトの各種予算が必要に応じて編成される。

6. ユーザ側の経常予算に対する影響が見積られ、必要があれば調整をとるための妥当な計画が立てられる。

B. 最終成果

1. プロジェクトの組織構成および人選。

2. プロジェクト全体の概略的作業計画。

3. 第4フェーズの詳細作業計画。

4. 各種予算。

C. ガイドライン

1. 「計画」フェーズは「プロジェクト計画点検」が終了する以前には実施されない。これは、どのようなプロジェクトについても、それが承認されるまでの出費をなるだけ抑えておくためである。

2. ここで行なわれる全面的な計画作業は時間を要するものであることが了解される。それに加え、各フェーズにおいても計画作業が行なわれなければならないため、長期的計画は変更の余地を有するものである。

第4フェーズ　研究・分析・調査

A. 主要ステップ

1. 現在のシステム、組織あるいは問題を研究し、これを文書化する。

(a) 管理者、業務担当者その他の職員からデータを収集する。

(b) 統計、文書類、手続、ファイルおよび報告書のサンプルを収集す

る。

(c) 関連文献その他の情報源を点検する。

(d) ユーザと協力して、調査もれはないか、結果は正確かを確かめる。

2. 分析

(a) ユーザの役に立つため、あるいは問題を解決するために、どのような情報が必要かを明らかにする。ここにおいて、プロジェクト担当者は、プロジェクトの最終成果は何かに関する最初の規定を与える。

(b) 上記2(a)を達成するためには何が必要かを明らかにする。例えばDPプロジェクトにおいては、この作業はシステム要件の決定と呼ばれるかも知れない。新システムに必要なインプットおよびプロセスを決定するのはこの時点においてである。

(c) ステップ2(a)、2(b)の遂行にかかわりのある主要な論理および方針を評価し要約する。

(d) 2(a)、2(c)の結果は、ユーザ側管理者と共にこれを点検して、その正しさについて同意を得るようにしなければならない。

3. 予備設計

(a) プロジェクトをいくつかの部分に分割する。このようなことは、プロジェクトが大きすぎて、全体を一つのチームで扱うのは無理だという場合必要になるであろう。プロジェクト内部での相互調整が欠如したりしないよう十分注意しなければならない。

(b) 上記2(a)、2(b)で述べられた要件を満たす予備的代替案の設計を行なう。

(c) 設計された各代替案毎のコスト、利益、スケジュールを更新する。

(d) あらゆる面から見て、諸要件を満たしていると思われる予備設計案を選び出す。

(e) 管理者の点検と承認を受けるため、設計された諸代替案および推薦案を提出する。複数の設計案を提出する理由は、管理者の側で、プロジェクト達成の難易度と共に、追加的節約、一層の弾力性、早期実施、より以上の顧客サービスその他の利益を得るため、どのような代償を払えるかを判断できるようにするためである。

4. 計画

(a) 選択された代替予備設計案にもとづき、設計フェーズのための詳細作業計画が立てられなければならない。

(b) プロジェクトのコストおよび人員計画は点検され、必要に応じて変更されなければならない。

(c) ここでの計画作業の結果は、管理者に提出し、その承認を受けなければならない。

B. 最終成果

1. 問題の確定

2. 予備設計代替案、そのうちの推薦案。

3. 必要に応じ修正された範囲説明書および全体作業計画。

4. 「設計フェーズ」の詳細作業計画。

C. ガイドライン

1. いくつかのプロジェクトにおいては、現行システムの研究が要求され、したがってまた、第4フェーズ、ステップ1の焦点は、新システムが満足すべき現存の諸要件の研究というところに置かれるかもしれない。この場合の研究の結果は、絶えずユーザと点検し、その正確性を確かめなければならない。この問題に関する決定は「計画フェーズ」期間中になされるべきである。

2. 「システム」という言葉は広い意味で用いられており、それは、組織上のシステムでも、ＤＰシステムでも、あるいはマニュアル　システムでもかまわない。

3. 本フェーズの予備設計においては、プロジェクトの分割がプロジェクト全体を損うことのないよう注意しなければならない。分割は、細かすぎたり、あまり早く手をつけたりしないようにすべきである。分割が認められるのは、明らかに作業の大きな切れ目となっているレベルにおいてのみであって、全体を分解させてしまうようなものであってはならない。

4. 本フェーズ全体を通じてユーザ側の関与が必要とされる。

第５フェーズ　設計

A. 主要ステップ

1. 前フェーズで選択された予備設計案に関して明確化されていない点、疑問点は徹底的に究明されなければならない。

2. 問題解決に必要な新組織、方策あるいはシステムとはどのようなものであるかに関する明細を作り上げる。そのなかには、プロジェクトのタイプによって以下の事項が含まれよう。

(a) 組織問題の研究の場合には、新組織の概念的表示

(b) ＤＰプロジェクトの場合には、システムと各部のフローの設計、システム、ファイル内容の決定、インプット　アウトプット　フォーマットの設計、プロセス要件の決定等

(c) コンサルタント的プロジェクトの場合は、ＯＲモデルあるいは意思決定用の選択肢を提供するために必要なツールの明細

(d) どの種のプロジェクトの場合も、書式および手続の設計は、プロジェクトのこの時点で規定される細目に及ぼすことになろう。

3. 開発中の解決策を時期を遅らせて問題領域に導入できるようにするために必要となる調整手段に対して、この時点で考慮を払っておくことが必要となるかもしれない。このような例としては人事の調整、あるいは新しいファイルが完成した時それを使って新システムのオペレーションができるようにするためのファイル　コンバージョンの準備などがあげられる。

4. 必要であれば、解決策のテスト計画を策定しなければならない。

5. 導入および監査の予備的計画を作成する。

6. ＤＰプロジェクトの場合、プロジェクト担当者は、この時点までにハードウェア、ソフトウェアの予備的仕様を作成すべきである。

7. この時までに判明した事実にもとづいて、コスト、利益ならびにスケジュールを更新する。

8. この時点で開発フェーズの詳細計画を立てるべきであり、それには、詳細作業計画および、もし必要であれば、関係のある範囲説明書の修正が含まれていなければならない。プロジェクトの現時点以降の部分に関する計画の点検も、ここに含まれていなければならない。

9. 「設計フェーズ」の結果は、システム側とユーザ側の管理者の点検、承認およびサインを受けるため、報告書にまとめ、これを提出しなければならない。

B. 最終成果

1. 解決策の確定的設計。

2. テスト計画（妥当する場合）。

3. 「導入」および「監査フェーズ」のための予備的計画。

4. ハードウェア、ソフトウェアの予備的仕様書（妥当する場合）。

5. 必要に応じて修正された範囲説明書および全体作業計画。

6. 「開発フェーズ」のための詳細作業計画。

C. ガイドライン

1. プロジェクトの最終的成功のためには、他のフェーズとならんで本フェーズ全体を通じてのユーザの関与が極めて重要であることを忘れてはならない。ユーザ側の関与が、たぶん最も重要なのはこのフェーズにおいてである。

2. 「設計フェーズ」の結果が提出され、その設計が受け入れられた場合、設計の明細、仕様は凍結され、「開発フェーズ」中いかなる変更も加えるべきではない。

3. ユーザならびにプロジェクトに関与する者は誰でも、次のフェーズでのシステム仕様の変更は、全プロジェクトの再評価を要し、おそらく全体の費用に重大な影響を及ぼすであろうことを理解しておくべきである。

第6フェーズ　開発

A. 主要ステップ

1. システムまたはその部分のため、ならびにファイル　コンバージョン作業のためのＤＰによる処理方法、あるいはラン　フローの概要を開発する。

(a) どのような機能が各プログラム内で演じられるかを決定する。

(b) 各プログラムのインプット　アウトプット　レコードの詳細レイ

アウトを開発する。

(c) 全解決策中の各分野におけるＤＰハードウェアの具体的使用形態を決定する。

(d) 開発されたＤＰ解決策の概要を、プロジェクトの管理者と共に点検する。

2. 全システムおよびファイル　コンバーションのための詳細プログラム仕様を開発する。

3. 特別のアプリケーション　パッケージを開発するかどうか、汎用プログラムを使うことができるかどうかを決定する。

4. プログラム　テストのための具体的要件を規定する。

5. プログラムのコーディング、デスク　チェックを行なう。

6. プログラムのコンパイル、テスト、デバッグ、ドキュメンテーションを行なう。

7. システムのある一部分を構成しているプログラム、相関関係にあるプログラムを一貫性あるものにする。

8. 書式、手続、コンバーション手続を含むユーザ用マニュアルを作成する。

9. 訓練プログラムを開発し実施する。

10. 詳細設計を行ない、組織変更をもたらすための諸ステップを開発する。

11. ユーザと協働し、ひとつの組織環境から別のものへ転換するためのテクニックを開発する。

12. 組織変更のため必要な一切の手続を開発する。

13. 組織に関する訓練の方法、テクニックを開発する。

14. 組織変更のため必要な訓練プログラムを実施する。

15. 範囲説明書および概略作業計画を点検し、必要に応じて変更する。

16. テスト　フェーズのための詳細作業計画を作成する。

B. 最終成果

1. テストおよび評価を受けることができるようになっている全面的に開発された問題の解決策。
2. 訓練プログラム。
3. 新システムのためのコンバージョン計画（妥当する場合）。
4. 然るべく修正された範囲説明書。
5. 「テストフェーズ」のための詳細作業計画。

C. ガイドライン

1. ＥＤＰプロジェクトにおいては、新装置を置く場所の計画と実際の導入が、極めて必要なステップとなるかもしれない。しかし、このステップのライフ　サイクル内の位置は、プロジェクトの特殊環境によって大きく違ってくる。この問題は、計画フェーズで考えておき、遅くともこの時点までには実行に移さなければならない。
2. このフェーズに到達した時点では、何をなすべきかが完全に明瞭になっていなければならないし、設計の不必要な変更は避けなければならない。追加的な設計変更やそれに類する「思いつき」については、後日別の企画において処理されることになろう。
3. ガイドライン２は、ビジネスの現実を無視しようというものではない。プロジェクトの基本目的にかかわる重大な設計変更については、プロジェクト内でとりあげなければならない。
4. 他のフェーズと同様、本フェーズ全体を通じてユーザ側からの関与は欠くことのできないものである。

第7フェーズ　テスト

A．主要ステップ

1. テスト　データを準備する。
2. ＤＰプロジェクトの場合には、システムあるいはその構成部分のプログラムをテストする。他のタイプのプロジェクトの場合は、用いられることになる手続あるいはテクニックをすべてテストする。
3. 開発したもののなかに特別のコンバージョン手続あるいはコンピュータ　プログラムがあれば、それらをテストする。
4. 必要な一切の手直しを行ない、適切なドキュメンテーションが行なわれるようにする。
5. ユーザと共に結果を点検し、それにもとづいて正式の承認、納得を得る。
6. 訓練を行なう。
7. 次の「実施　評価フェーズ」の作業計画を含む詳細計画を作成する。プロジェクトの範囲説明書、コストを然るべく修正する。

B．最終成果

1. 全面的にテストされ、必要な訂正を加えた解決策あるいは新作業方法。
2. ユーザによる解決策の承認。
3. 然るべく修正された範囲説明書。
4. 「実施　評価フェーズ」のための詳細作業計画。

C．ガイドライン

1. 人工的、歴史的あるいは生の情報を用いた色々な規模のテストが実施されることになろう。

2. テスト　フェーズは、プロジェクトのタイプ、その内部での高度性の程度によって相当違ったものとなろう。この違いは、どのようなことが必要となるかについて、計画フェーズで注意深く検討することにより対処することができよう。

3. このフェーズは、あらゆるプロジェクトに、何らかの形であてはまるものである。

第8フェーズ　実施・評価

A. 主要ステップ

1. 最初のステップは部分的な実施、あるいはパイロット　テストで、これにより、いままでの努力の成果が現実の生の環境で使いものになるかどうかを確かめる。この種のテストは、プロジェクトを管理する側が、生の条件でテストしないうちに全面的実施をするのは妥当でないと考えた場合に行なわれる。

2. 可能であれば、従来の方式で業務を続けながら、全面平行テストの準備をし、これを実施する。これにより、この重大なテスト　ポイントで解決策が破綻した時の「つっかい棒」が与えられる。この種のテストは、一定の状況においては採用できないかもしれない。

3. 上の結果にもとづいて、解決策あるいは新テクニック、またはシステム導入の全面的実施、始動に対するユーザの承認を受ける。

4. 全プロジェクトの文書化を完了する。

5. 一切の訓練を完了する。

6. もし、プロジェクトがＤＰプロジェクトであったり、あるいは日常のオペレーションに関与する外部のサービス　グループである場合、

ステップ１～５の結果は、そのグループの責任者の点検を受け、こうして全関係者が合意した時、開発されたシステムあるいは解決策が通常の業務のために採用される。

7. プロジェクト実施初期の結果を評価し、必要に応じて訂正を行なう。

8. 次の「事後監査フェーズ」のための詳細作業計画を立てる。

9. 旧システムがプロジェクトの成果によって置換えられる場合には、新システムへの転換を完了し、旧システムの使用を打切る。

B．最終成果

1. パイロット　テストの結果（妥当する場合）。

2. テスト結果に対するユーザの了承。

3. 完了した訓練。

4. 全面使用できる解決策。

5. 「監査フェーズ」のための詳細作業計画。

C．ガイドライン

1. いろいろな「思いつき」をプロジェクトに追加しようと作業を続けることは、特にこのフェーズでは避けなければならない。こういったことは、それらが経済的に見て妥当であれば、別のプロジェクトがメインテナンス作業において行なうことができるものである。

2. このフェーズを通じ、ユーザの関与および結果の見きわめが特に重要である。

第９フェーズ　事後監査

A．主要ステップ

1. 業績監査、これには以下の事項が含まれる。

－運営状態および能率

－コスト

－利　益

－ユーザおよび顧客の満足度

－保安およびバックアップ状況

－ドキュメンテーション

－ハードウェアの性能（使用している場合）

－手続の使用状況

2. 財務監査．これには以下の事項が含まれる。

－プロジェクトの全体的結果の一貫性

－監査証跡

－正確性

B．最終成果

1. 監査報告。

2. 勧告。

C．ガイドライン

1. 経理あるいは財務関係のプロジェクトにおいては、監査のためのテクニックおよびスタッフの使用は、必要に応じプロジェクトの内部で行なわれるべきである。

2. 「事後監査フェーズ」を担当する者は、第1～8フェースに関与したプロジェクト担当者であってはならない。

昭和48年3月
発行人 日本経営情報開発協会
東京都千代田区霞が関3－2－5
（霞が関ビル30階）
電 話（581）6401番
印 刷 株式会社 三 州 社
東京都港区芝大門1－1－21
電 話（433）1481(代)